ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT DRAFT

A MICRO-PENETROMETER FOR SNOW AND SOIL STRUCTURAL ANALYSIS

Team #12

CHENGHAO MO (cmo8@illinois.edu)
XING SHEN (xings2@illinois.edu)
ZHEYAN WU (zheyanw2@illinois.edu)
CHENXIAN MENG (cmeng10@illinois.edu)

May 8, 2024

Abstract

Micro-penetrometers serve as indispensable tools in various scientific and environmental applications for non-destructive analysis of soil and snow compositions and their structural relationships. However, existing micro-penetrometers encounter limitations such as mobility constraints and inefficiencies in batch measurement and data transmission. This report elaborates on the development process of our enhanced micro-penetrometer, undertaken as part of the ECE445 course. Our micro-penetrometer builds upon existing functionalities by incorporating horizontal motion along the x and y axes, facilitating enhanced maneuverability. Additionally, we have integrated Bluetooth technology for seamless data transmission, ensuring portability and convenience. This project represents a significant advancement in micro-penetrometer design, addressing key limitations and enhancing usability across scientific research and environmental forecasting domains.

Contents

1 Introduction

1.1 Problem and Solution Overview

In various fields such as agricultural production, archaeology, and disaster warning, the analysis of soil (sand, snow) is particularly important. The use of mechanical penetration can detect the bonding forces of soil (sand, snow) at different depths, and serve for subsequent data analysis. The existing analytical instruments have many problems, such as too large volume, limited application scenarios, insufficient operation convenience, and insufficient accuracy[1], [2]. We aim to develop a portable, easy-to-use device that delivers intuitive results and can adapt to various scenarios for users.

We envisioned designing a machine that could drive a rod with a force sensor into soil (sand, snow) and record the force it received in real time. After integrating the data records, the penetration force characteristics of the sample at different depths could be visually reflected by color. Furthermore, this machine could also achieve multiple sampling and analysis of small-scale ground samples without moving the device by changing the horizontal position of the rod.

The specific implementation method is as follows: The controller controls the movement of the mechanical structure. During the movement, the data collected by the force sensor can be transmitted to the computer in real time by Bluetooth, and stored in the computer. The data analysis and result presentation can be carried out through the supporting software on the computer. After a single sampling, the rod will be reset to the initial height. If necessary, the horizontal position of the rod will be changed and the sample will be taken again. In addition, in the case of Bluetooth connection, users can also control the machine by using Bluetooth devices.

Our solution is to design an mechanical product which can test the force when drugging into the soil with pressure sensor and move xyz axis automatically. Also, we need to design the control parts to control the motor movement and read the transient data of sensor. The data can be displayed on computer with Bluetooth block.

1.2 System Components

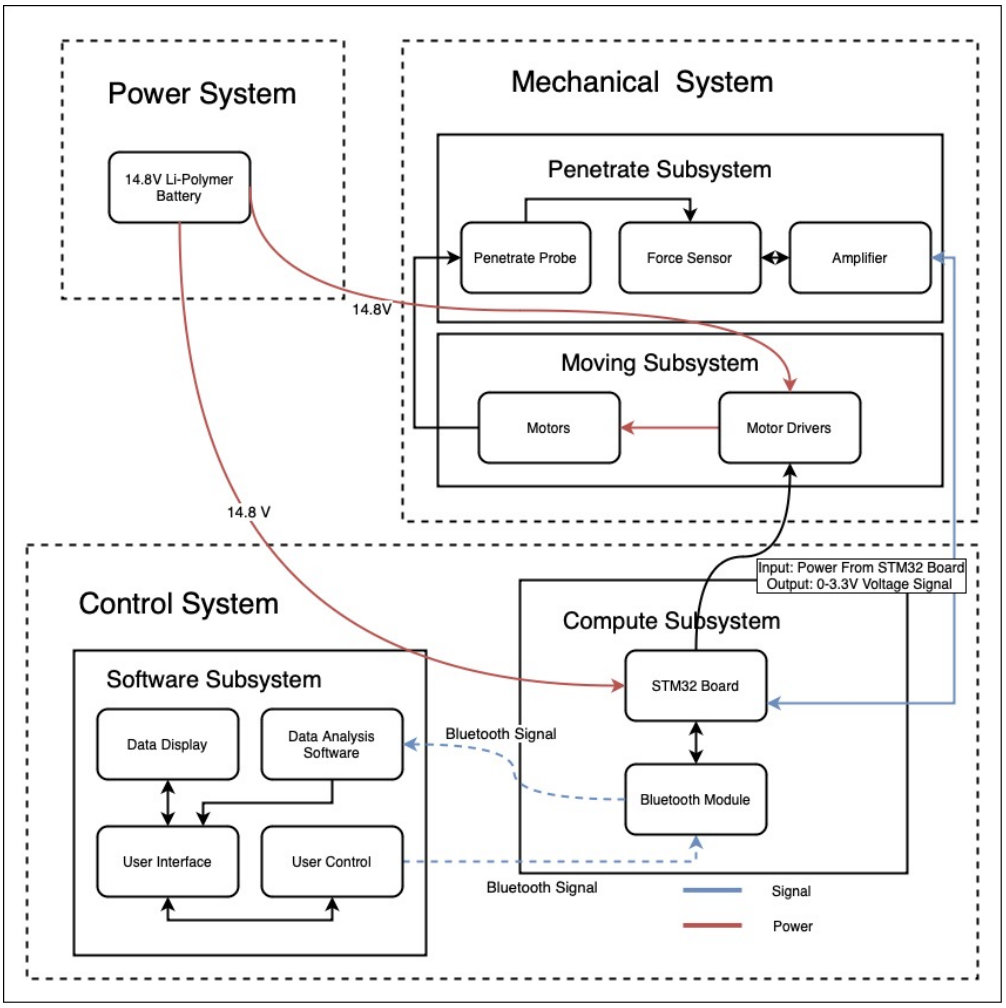


Figure 1: Block Diagram

The system is divided into several key blocks, shown in figure 1, each meticulously designed to handle specific functionalities within the overall operation:

- **Mechanical System:** Comprises three stepper motors; two manage horizontal movement across a 7x7 grid, and one is connected to a lead screw for precise vertical drilling.
- **Sensing and Data Acquisition:** Utilizes a force sensor to detect resistance as the probe penetrates different materials. Data are transmitted via Bluetooth to the host computer where it is processed and visualized.
- **Control Unit:** Features an embedded STM32 microcontroller that processes sensor inputs, manages motor operations, and communicates via Bluetooth with the host computer.

- **Control Program:** Includes a user interface that allows scanning for available devices and another interface for sending commands, receiving data, and plotting results..

1.3 Performance Requirements

The performance requirements defined in the final version of our proposal include:

- Maximum penetration depth of 50 cm.
- The probe can move horizontally, with a minimum displacement of at least 30cm along both the x and y axes.
- The software can record information for all detection points, including depth and the magnitude of pressure applied.
- Users can control the entire system via Bluetooth.
- The error of the horizontal motor should be controlled within 5%, while the error of the vertical motor should be controlled within 2%.
- The software will be able to provide a 3D visualization of the data results, allowing users to have a more intuitive understanding of the detected terrain.
- Battery life sufficient to support at least 50 uses on a single charge.

1.4 Design Modifications

Throughout the development process, several significant modifications were made to enhance the performance of the system. These changes were driven by initial testing outcomes and feedback, leading to improvements in sensor integration and motor control mechanisms.

- **Sensor Integration:** Initially, the force sensor was directly connected to the data acquisition system. To improve the accuracy and reduce noise interference, we integrated the force sensor with a charge amplifier. This setup not only enhanced the signal-to-noise ratio but also improved the overall measurement accuracy. Additionally, modifications were made to the mounting of the force sensor to further minimize noise and increase the precision of the data captured.
- **Motor Control:** The control of the motors was initially based on basic algorithms which did not account for the non-linearities in motor response. To address this, we implemented a Pulse Width Modulation (PWM) algorithm to refine the control over the motor's motion angles. This allowed for precise control of the motor's position and, consequently, the depth to which the probe could penetrate. The revised algorithm also facilitated an accurate transformation of angular movement into specific depth measurements, significantly improving the responsiveness and accuracy of the probe's movements.

2 Design

2.1 Design procedure

2.1.1 Structure Design

In terms of mechanical structure design, the first is the overall design of the structure. Since we need to control the movement of the detection rod in three directions xyz, we came up with a structure similar to a 3D printer, with independent motors in each direction to control its movement in their respective directions. The first idea we came up with was similar to a cube framework, but soon we found a better structure to do this. The new structure uses less material and is also smaller, more in line with our requirements for portability and light weight. After the structure is roughly determined, the choice of transmission mechanism is also very important. At present, the more popular three-axis transmission mainly has two ways of screw structure and belt transmission, the former has the advantage of more accurate displacement and greater force, and the latter has the advantage of faster movement, lighter weight and cheaper price. Finally, we used a screw structure on the z axis and a belt structure on the xy axis. Because the Z-axis is responsible for pushing the detection rod into the soil while ensuring uniform movement, while the movement of the xy axis has less demand for accuracy and force. After comprehensive consideration of cost, accuracy and other factors, we finally decided to buy the z axis screw structure online, and the xy axis belt structure is assembled by ourselves.

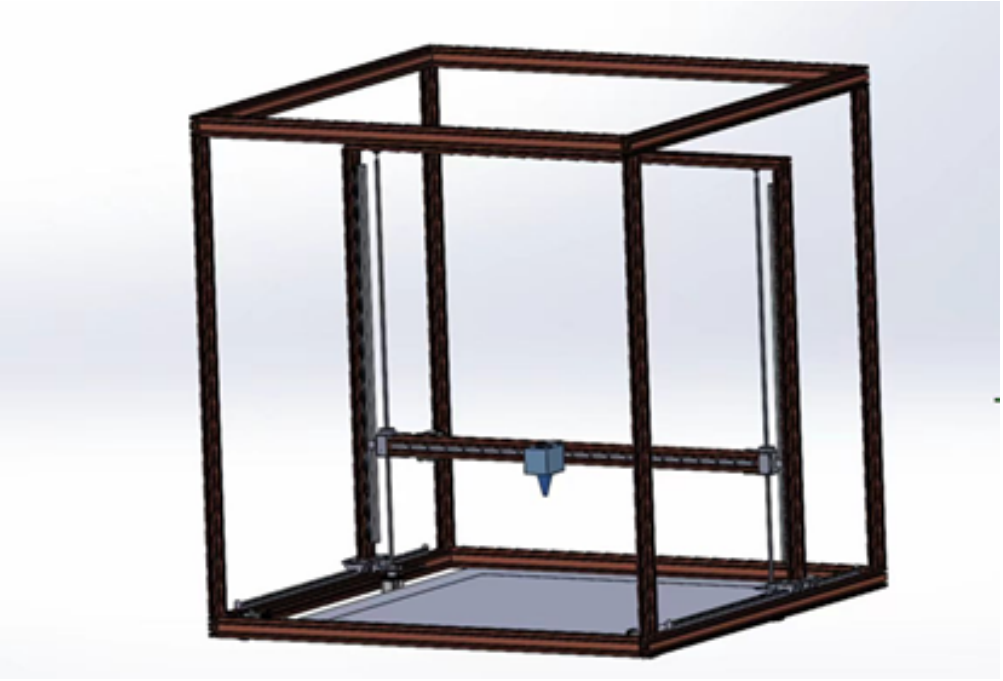


Figure 2: First Design

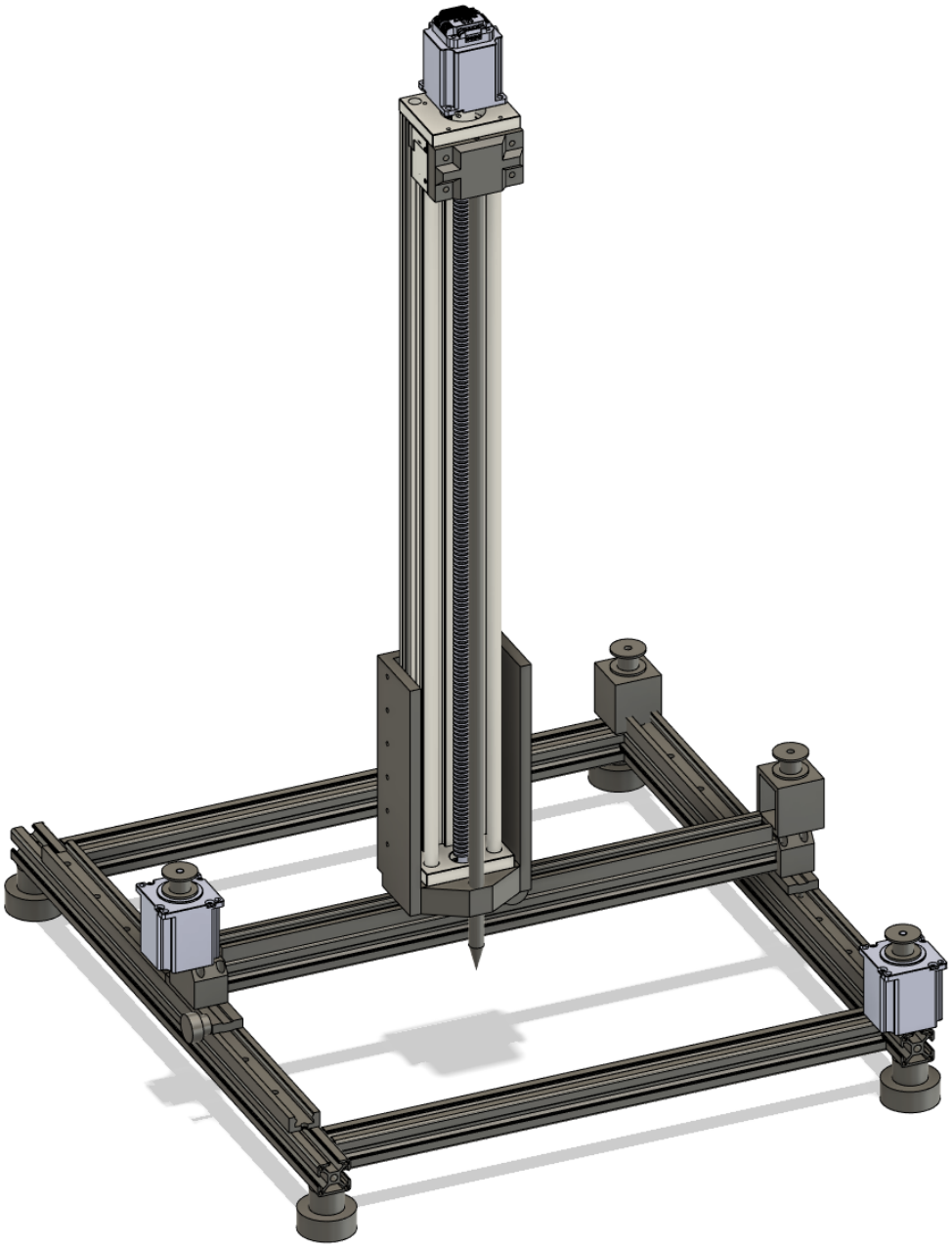


Figure 3: Final Design

2.1.2 Connection Schemes

There are some connection schemes in the assembly process. For our design, the main options are the use of glue and screw fixation. In our design, most of them are fixed by screws. Compared with the direct use of glue, its advantage is that it is easy to disassemble, which is conducive to our modification, and the strength of the connection is large, which will not be damaged. However, there are also disadvantages, in addition to the connection between aluminum profiles can use corner code and ladder nuts, the con-

nection between other parts require us to design and print 3D printed parts, and design screw holes on the 3D printed parts for connection. So in our design, all the connections in the main structure are fixed with corner codes or 3D printed connectors and screws. Of course, we also use glue to fix, such as the fixing of the motor is the use of glue. There are two main reasons for this, one is because the screw hole reserved on the motor above the aluminum profile is back to the aluminum profile, which means that we need a larger connector to wrap the upper part of the motor in order to use the screw connection, adding a connector in the limited space will affect the normal operation of other parts; In addition, because the motor is subjected to less external force, the strength of bonding using 502 glue is enough, and the glue is more convenient.

2.1.3 Force Sensor

In the choice of force sensor, we originally intended to use the most conventional laboratory force sensor, but this idea was quickly rejected - the laboratory force sensor is too large and follows the pole into the soil.Therefore, we wanted to use the ordinary piezoelectric sensor module, but found that it has two shortcomings: first, the module is wired from the side, which will lead to the actual need for more space than its own area, because it needs to ensure that the wire is passed through the rod under the premise of the sensor being flat; Second, because the sensor itself can not work alone, it needs to cooperate with the design of the external structure to complete the detection of the force on the tip, which requires the external structure to be more fine to ensure the accuracy of the measurement results, which is more difficult to achieve 3D printing. So we found the sensor we are using now, it is in line with the characteristics of round, small size, the bottom line can also meet the design needs, is undoubtedly the best choice.

2.2 Design details

2.2.1 Force Sensor

Input: : Physical force

Output: Voltage change of signal

The force sensor has a built-in Force Sensitive Resistor (FSR) whose resistance varies according to the applied pressure. The force sensor here, with a 0-300N range and a sensitivity resolution of 1.0±10% mV/V, is a strain gauge-based transducer designed to convert applied force into a measurable electrical signal. When interfaced with an STM32F407IGT6 microcontroller, the sensor's excitation wires are connected to a stable 5V supply and ground, while the signal wires are linked to the microcontroller's ADC inputs to facilitate precise data acquisition. This setup enables the conversion of the mechanical force into digital values, which the STM32 controller can process and interpret. Calibration is imperative to correlate the ADC readings to actual force measurements accurately, considering the sensor's specified sensitivity and ensuring that the 10% tolerance is accounted for in the measurement system. This integration not only provides a reliable method to measure forces within its capacity but also allows for real-time monitoring and

analysis when coupled with the microcontroller's computational capabilities, making it a suitable choice for a wide range of applications demanding force quantification. The figure 4 shows the schematic diagram of the force sensor.

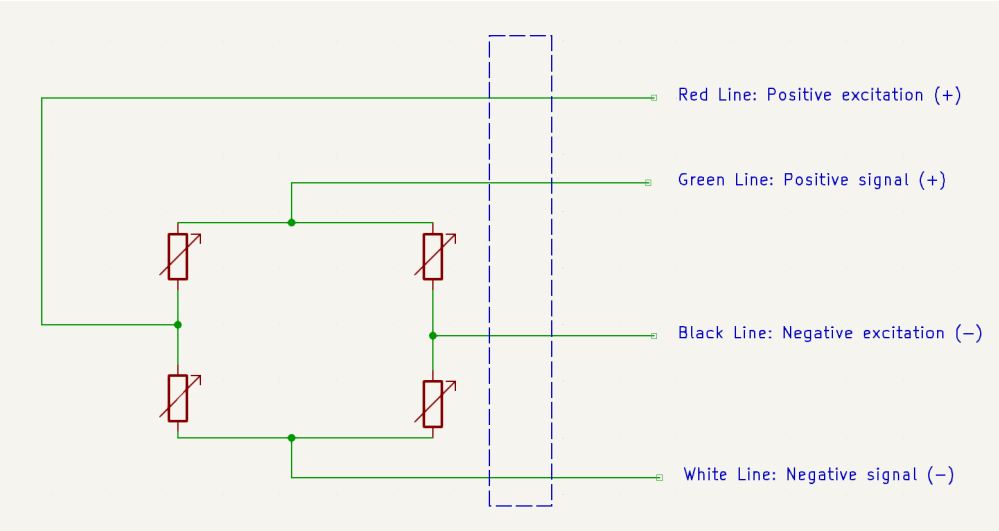


Figure 4: Force Sensor Schematic Diagram

2.2.2 Voltage Amplifier

Input: Positive Signal (Force Sensor), Negative Signal (Force Sensor), Power Supply (24V)

Output: Amplified Voltage (+), Amplified Voltage (-), Power Supply for Sensor

The voltage amplifier is a critical component in our system, designed to enhance the low output signal from the force sensor for accurate digital interpretation by the STM32F407 microcontroller. Given that the force sensor outputs a very low signal of approximately 1 mV, the amplifier's role is to boost this signal to a more usable level. The selected charge amplifier outputs a voltage in the range of 0-5V, which corresponds to the force sensor's measurement range up to 30 kg. This linear relationship between the output voltage and the force ensures that 5V represents the maximum force of 30 kg, allowing for precise and straightforward data interpretation.

Additionally, the amplifier is powered by a 24V supply, ensuring it has sufficient power to operate effectively. Despite this high input voltage, the amplifier is designed to output the necessary 0-5V signal, making it compatible with the microcontroller's ADC, which has a maximum input voltage of 3.3V. This careful matching of components ensures that the amplified signal remains within the optimal range for the STM32, facilitating accurate and reliable force measurements. Integrating the amplifier with the microcontroller's onboard voltage output ports not only streamlines the design by eliminating the need for an additional power source but also ensures the system is both efficient and compact, reducing the overall complexity of the design.

2.2.3 Motor Unit

High-performance stepper motor driver (ATK-PD5050S)

The ATK-PD5050S module [3] is a versatile and rugged stepper motor driver that enables motors to dig down at a steady speed. The module operates from a 12 to 50V DC supply voltage range and has an output current of up to 5.0A, effectively driving a two-phase hybrid stepper motor at a constant speed. It boasts cutting-edge features including high-resolution microstepping for detailed motion control, load-based power optimisation for energy savings, and a low-resonance chopper algorithm that minimises vibration for improved motion accuracy. Integrating the ATK- PD5050S with an STM32F407IGT6 microcontroller typically involves connecting the drive's power input to the microcontroller's controlled voltage output, aligning the motor connections with the drive's terminals, and establishing communication for pulse, direction, and enable functions via optically isolated control signals. This integration will facilitate sophisticated motor control through the microcontroller's firmware, which generates PWM signals to control motor direction and to engage or disengage the motor as required by the application.

The figure 5 shows the schematic diagram of the motor, and the table 1 shows the motor pin descriptions.

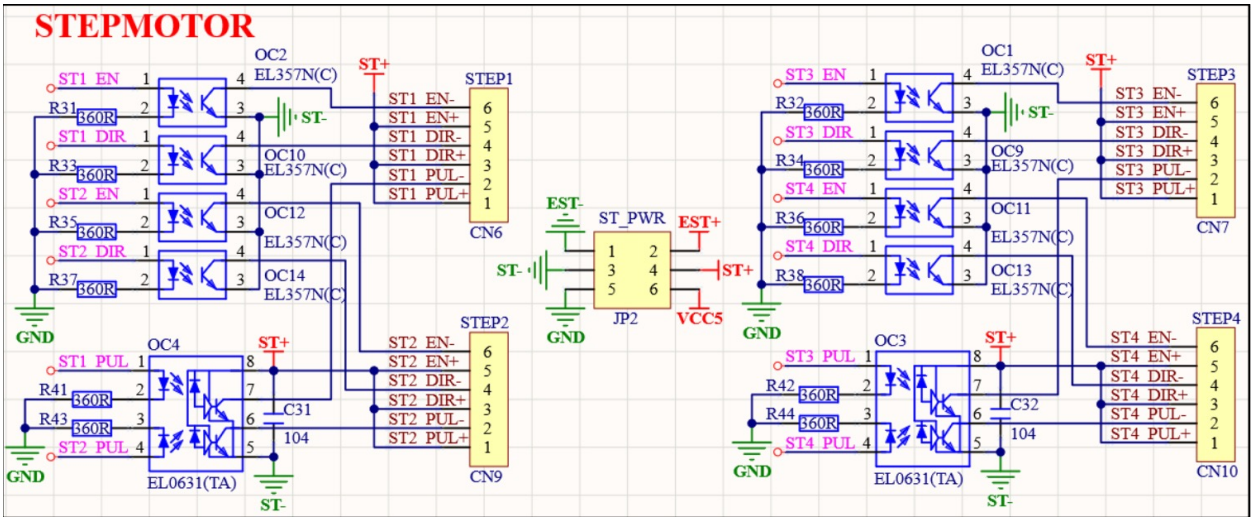


Figure 5: Motor Unit Schematic Diagram

Pin	Name	Description
ENA-	Enable negative	When the negative enable signal is effective, the internal logic signal is valid. When the signal is invalid, the internal logic is suspended, and the output is low impedance, high level, and the internal clock is stopped.
ENA+ (5V)	Enable positive (5V)	When the positive enable signal is effective, the internal logic signal is valid. When the signal is invalid, the internal clock is suspended, and the output is high impedance, high level.
DIR-	Direction negative	When the negative direction signal is effective, the internal logic determines the direction, and when the signal is invalid, the direction is determined externally.
DIR+ (5V)	Direction positive (5V)	When the positive direction signal is effective, the internal logic determines the direction.
PUL-	Pulse negative	When the negative pulse signal is effective, the internal logic is triggered, and when the signal is invalid, the external high level maintains the current state.
PUL+ (5V)	Pulse positive (5V)	When the positive pulse signal is effective, the internal logic is triggered, and the speed of the pulse is 200kHz.

Table 1: Motor Driver Pin Descriptions

We hope that this accuracy should be large than 95%. On x and y axis, the large accuracy means the probe can be control to the right place. Since we want to detect several place in a $50cm * 50cm$ square, we should keep every position accurate. Also, for the z axis, we also hope we can generate a diagram with force and distance, which means the motor should be accurate enough to represent its position.

Besides, we should also test the accuracy for different motors. The accuracy for each motor may be different. Which means when we setting parameters and controlling the motors, we should assign different parameters to different motors. So this step should be repeat several times for every motors.

2.2.4 Microcontroller (STM32F407)

Input: 5V(USB) or DC6V 24V(DC005), voltage signal from sensor, data from motor driver
Output: command to motor driver, data of force sensor to Bluetooth Module

The microcontroller (ATK-DMF407)[4] serves as the central hub for interfacing with the

Bluetooth module (ATK-MW579), the force sensor (DYMH-106), and the motor unit (ATK-PD5050S). It connects to the Bluetooth module utilizing TX and RX pins for bi-directional data exchange, allowing for wireless communication with other devices using UART. The force sensor is linked to an analog GPIO pin on the microcontroller, enabling the measurement of variable voltages that correspond to physical forces applied to the sensor. This analog signal is then digitized by the microcontroller for processing. For the motor unit, the microcontroller uses digital GPIO pins to send control signals (enable, direction, and pulse) to the stepper motor driver, dictating the motor's operational state, direction, and speed. This intricate network of connections between the microcontroller and the various components facilitates a seamless integration, enabling the microcontroller to collect data, process inputs, and control outputs, thus forming the backbone of a sophisticated sensor and control system. Figure 6 shows the STM32 Voltage Amplifier and Force Sensor Schematic Diagram.

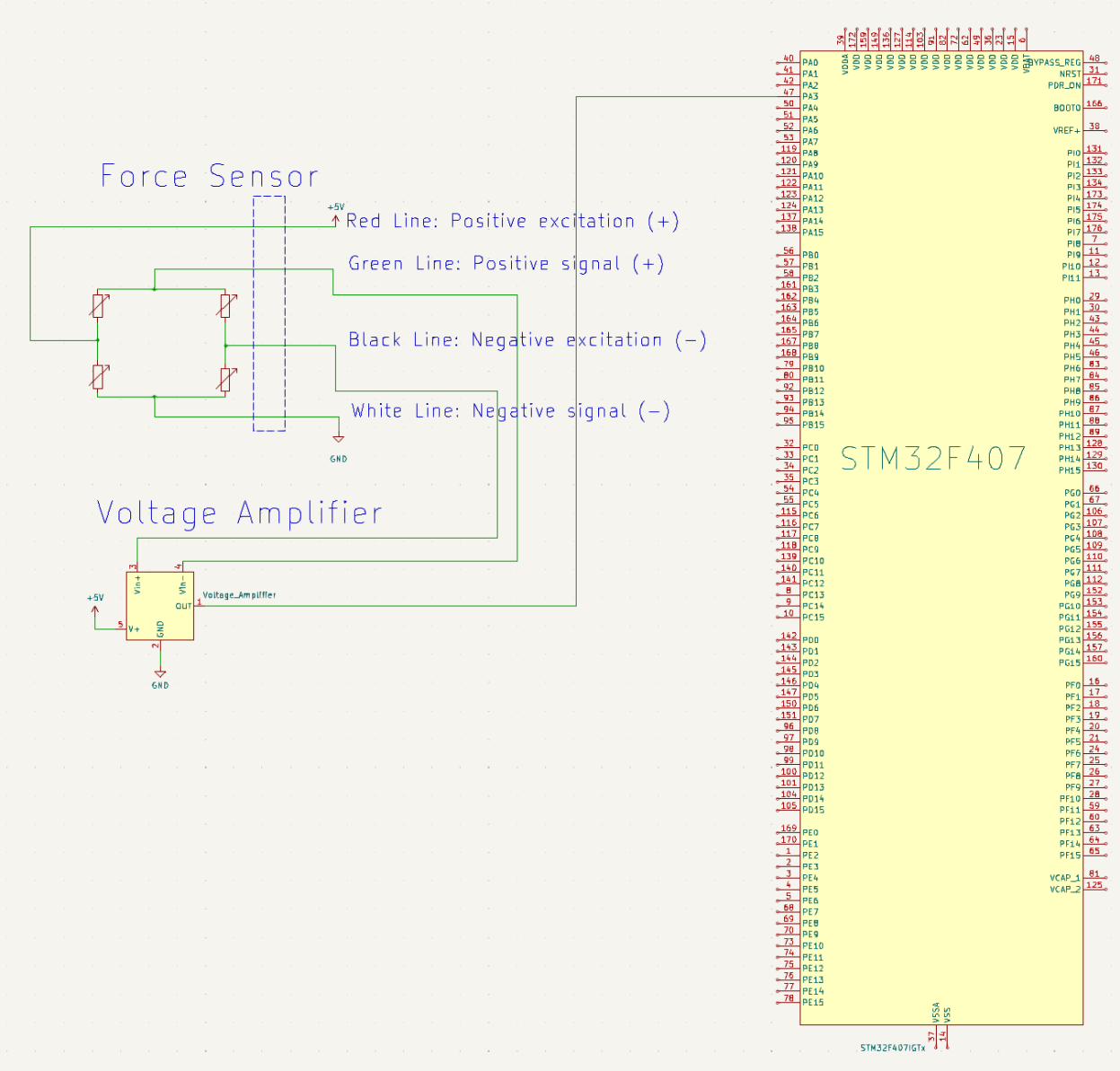


Figure 6: STM32 Voltage Amplifier and Force Sensor Schematic Diagram

Component Interface	ATK-MW579 Pin	ATK-PD5050S Interface	F407 Corresponding Interface
Power Supply (VCC)	VCC	-	5V
Ground (GND)	GND	-	GND
UART Transmit (TXD)	TXD	-	PB11
UART Receive (RXD)	RXD	-	PB10
Status Indicator	STA	-	PF6
Wake Up (WKUP)	WKUP	-	PC0
Enable Negative	-	ENA-	PF15
Enable Positive	-	ENA+	ST+ (Supply Voltage Positive)
Direction Negative	-	DIR-	PF14
Direction Positive	-	DIR+	ST+ (Supply Voltage Positive)
Pulse Negative	-	PUL-	PI5
Pulse Positive	-	PUL+	ST+ (Supply Voltage Positive)

Table 2: Microcontroller Pin Interface Mapping

2.2.5 Bluetooth Module (ATK-MW579)

Input: 3.3V 5V power supply and pin connection with microcontroller Output: data transmitted to host device (Bluetooth protocol)

The Bluetooth Module modulates 1Mbps enhanced data rate with complete 2.4GHz radio[5].

Pin	Name	Description
1, 3, 9, 19, 20	GND	Ground
2	ANT	Antenna
4	SLEEP	Sleep Mode
5	WKUP	Wake Up, Active High
6	RELOAD	Reload Counter
7	RXD	UART Receive (RX) Pin
8	TXD	UART Transmit (TX) Pin
10	3V3	Power Supply (3.3V–5V)
11–15, 18	NC	Not Connected
16	LINK	Connection Status Indicator (Active Low: Connection, Active High: No connection, Flashing: Transmitting)
17	RST	Reset (Active Low)

Table 3: Pin Descriptions of the Bluetooth Module

Connection with STM32 dev board.

ATK-MW579 Pin	F407 Corresponding Interface
VCC	5V
GND	GND
TXD	PB11
RXD	PB10
STA	PF6
WKUP	PC0

Table 4: ATK-MW579 to F407 Interface Mapping

2.2.6 Control Program

The control applet is a crucial component of the micro-penetrometer system, designed to facilitate seamless interaction between the user and the device through Bluetooth communication. It combines user-friendly interface design with robust functionality to control device operations and visualize data effectively.

System Architecture and User Interface The applet is structured into two main pages as shown in figure 7, each serving distinct functions:

1. **Device Scan Page:** Allows users to scan for available Bluetooth devices, displaying each device with its Bluetooth address. Users can select and establish a connection to the micro-penetrometer from this page.
2. **Command and Control Page:** Activated post-connection, this page includes:
    - **Grid Control:** A 7x7 interactive grid where users can select coordinates for the horizontal movement of the probe, sending precise movement instructions to the system.
    - **Operational Commands:** 'Start' and 'Return' buttons to begin and reverse the drilling operation, simplifying probe management.
    - **Data Visualization:** Features for plotting and analyzing data, such as an overview plot of force values across various positions and depths, and options to select specific coordinates for detailed analysis.

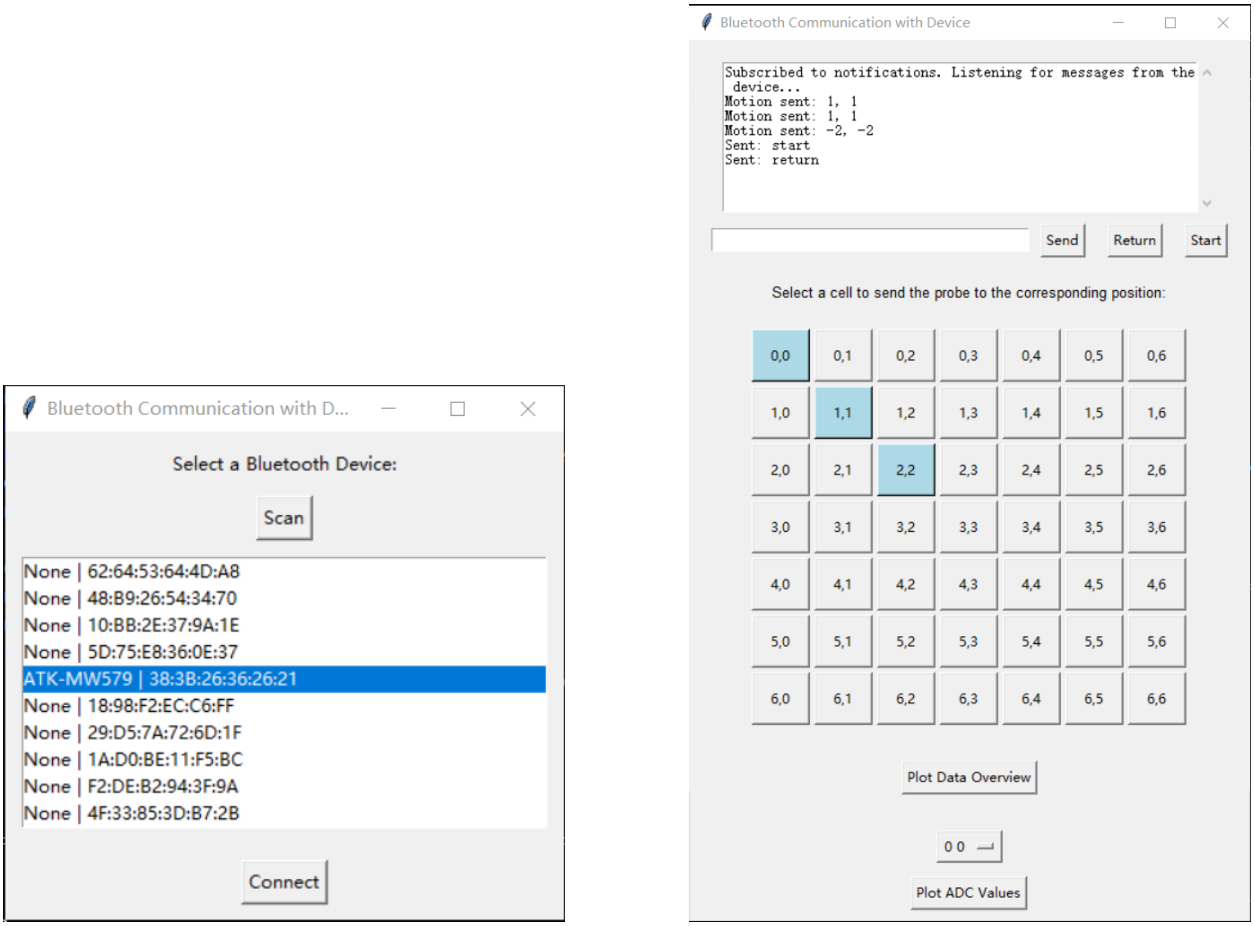


Figure 7: Left: Device Scan Page. Right: Command and Control Page.

Integration and Functionality The program communicates directly with the STM32 microcontroller via Bluetooth. It sends encoded command strings to the microcontroller, which interprets these and controls the motors accordingly. Force data collected by the system is sent back to the applet for processing and visualization:

- **Data Flow:** The program encodes operational commands as strings, which are decoded by the STM32 to execute motor actions. Feedback from the force sensor is transmitted back to the applet and displayed graphically.
- **Graphical Representation:** Real-time graphical outputs generated within the applet illustrate the force dynamics relative to probe depth and position, using python matplotlib.pyplot for data visualization.

Design Considerations To enhance the system's usability and extend its analytical capabilities, several key design considerations were implemented:

- **Data Storage and Analysis:** All data collected during probe operations are stored in a SQLite database, which offers robust data management without the limitations of the STM32's onboard storage capacity. This approach allows for extensive data accumulation without concerns about space constraints. The use of SQLite facilitates complex data queries and supports extensive analysis post-collection. Further, data visualization is performed using Python packages, which provide powerful tools for generating insightful graphical representations from the stored data.
- **Plotting Methods:** The control applet incorporates two primary methods for data visualization:
    1. **Overall Plotting:** This method provides a comprehensive overview of all collected data, allowing users to quickly assess and analyze the force values across all tested positions and depths. It is particularly useful for identifying patterns and anomalies across the entire data set. The figure 8 shows the overall plotting.
    2. **Position-Specific Plotting:** Enables detailed visualization of data at specific grid coordinates. This feature is essential for in-depth analysis of the probe's performance at particular locations, facilitating detailed assessments that may inform adjustments or further targeted investigations.The figure 9 shows the overall plotting.

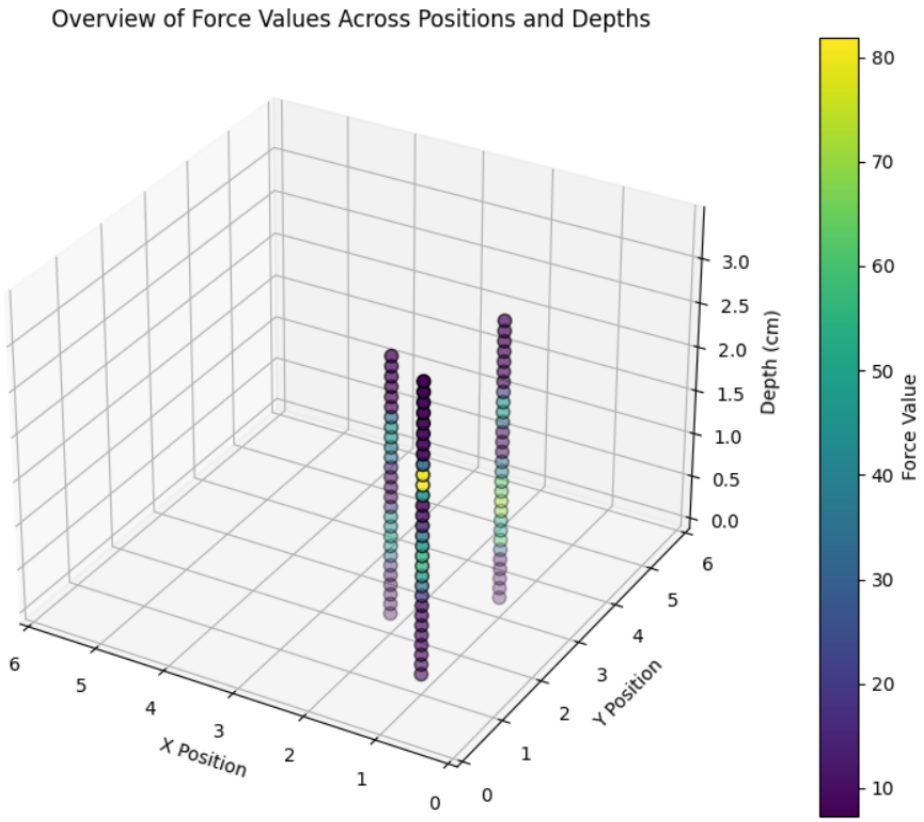
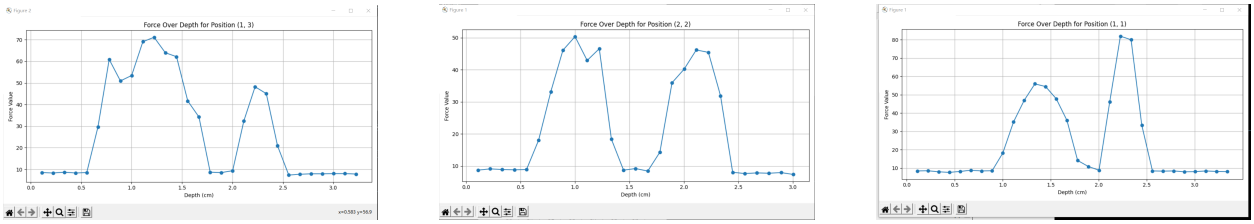


Figure 8: Overall Plotting



(a) Position (1,3) Plot (b) Position (2,2) Plot (c) Position (1,1) Plot

Figure 9: Position-Specific Plotting

These considerations are crucial for ensuring the system not only performs data collection efficiently but also supports detailed analysis and easy access to historical data for longitudinal studies or iterative testing scenarios. The integration of robust data handling and versatile visualization options enhances the overall functionality and user experience of the micro-penetrometer system.

2.2.7 Connector Designs

Here are some important connectors we designed. They are finally processed in the form of 3D printing.

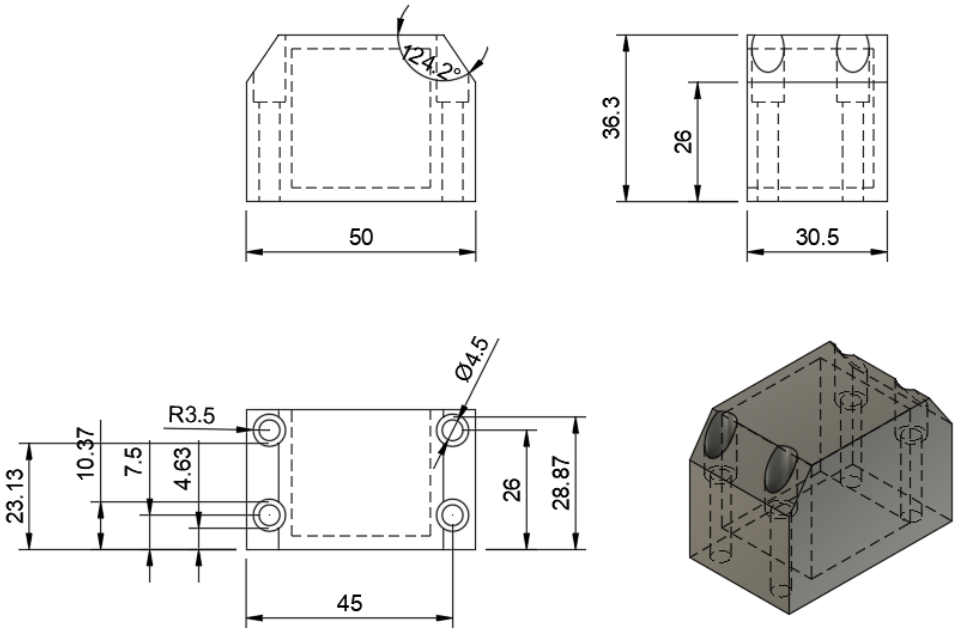


Figure 10: Connector 1

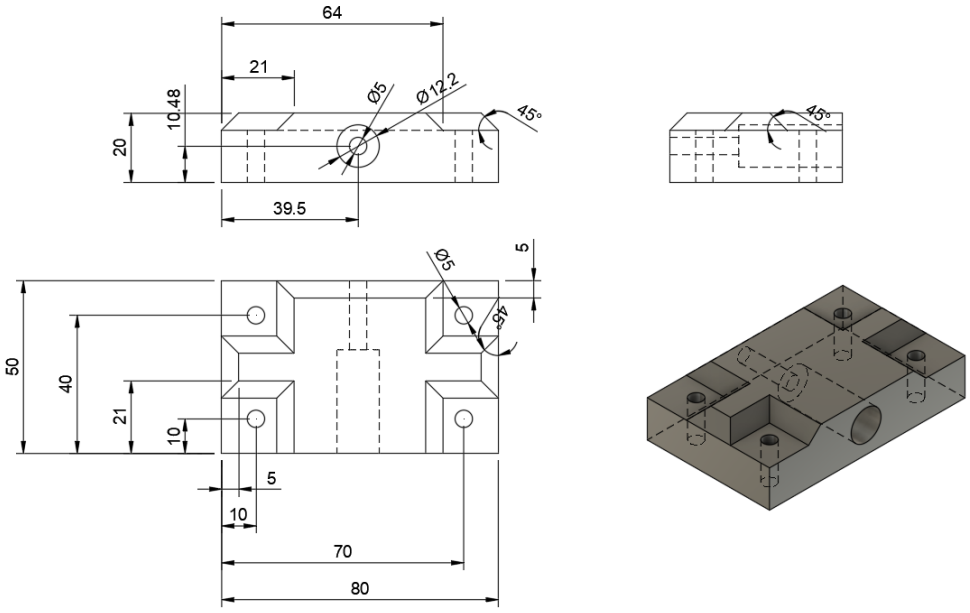


Figure 11: Connector 2

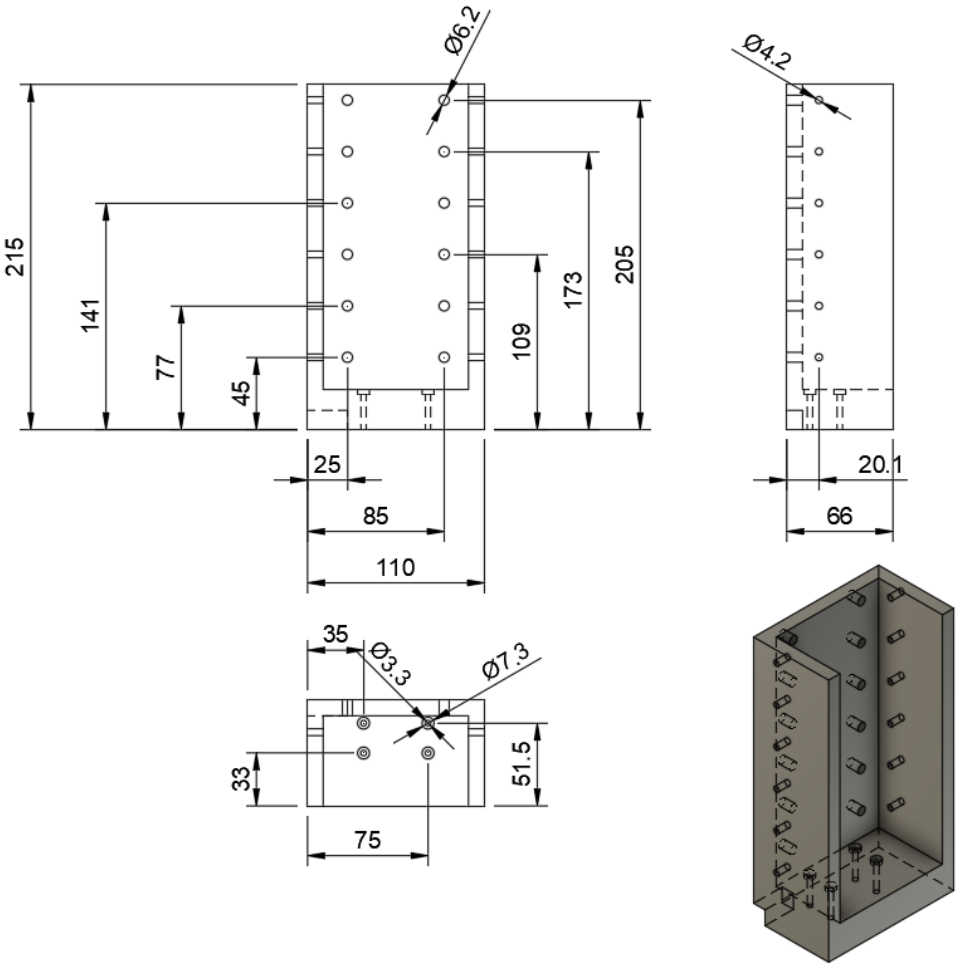


Figure 12: Connector 3

3 Verification

3.1 Force Sensor

The force sensor is the most critical component in data acquisition for this project. The verification method used for this component consists mainly of two steps: validating the linear relationship between the output voltage of the sensor and the applied force, and calculating the accuracy of proportional relationship between the force exerted on the sensor and its output voltage. Table 9 shows the requirement and verification table for the force sensor.

3.1.1 Verification Process - Linearity

Validation of the linear relationship of the force sensor is based on the premise that when the sensor receives the same magnitude of tension or pressure, its output voltage remains constant. Our validation method relies on a precise laboratory force sensor. We conducted experiments by applying tension and simultaneously recording the voltage output of the sensor as well as the force measured by the laboratory force sensor, then plotted the data. Figure 13 shows the verification for linearity.

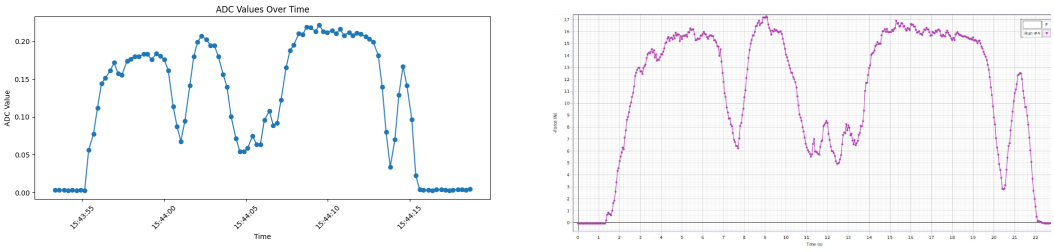


Figure 13: The Verification for Linearity

The left of figure 13 displays the voltage signals received by the microcontroller from the sensor, which have been amplified by an amplifier. The x-axis represents time, and the y-axis represents the voltage magnitude. The right of figure 13 shows the force signals received by the laboratory's force sensor, with the x-axis also representing time and the y-axis representing force magnitude. Comparing the two images, we can observe that their trends are essentially identical, with differences primarily attributable to variations in sampling frequencies between the two sensors. Given that the force displayed by the right-side sensor exhibits a linear relationship, it is not difficult to demonstrate that the force exerted by our sensor is also linearly correlated with voltage magnitude.

3.1.2 Verification Process - Accuracy of Proportional Relationship

The validation of the proportional relationship of the force sensor involves ensuring that the force measured by the sensor corresponds accurately to the voltage output. The following steps outline the verification process:

First, the experiment setup requires placing the force sensor in contact with the scale, ensuring it is perpendicular to the plane of the scale. This proper alignment is crucial for accurate measurement and consistency in results.

Next, the measurement recording involves several steps. Initially, record the degree of the force sensor in terms of voltage. Following this, calculate the force using the formula (1):

$$\text{Force} = \text{Voltage} \times \frac{30\,(\text{force sensor range}) \times 9.81}{5\,(\text{charge amplifier range})} \tag{1}$$

Simultaneously, record the force values measured by the scale. This dual recording ensures that the force values calculated from the voltage output of the force sensor can be compared against the actual force values measured by the scale, thus validating the accuracy and proportionality of the force sensor. The figure 14 shows the process.

Figure 14: The scale used to measure the voltage output of the force sensor in contact with it.

The comparison graph of standard force versus detected force by the force sensor (see Figure 15) shows a high degree of correlation between the two measurements. The nearly overlapping lines suggest that the force sensor accurately measures the applied force across different magnitudes. This accuracy is further supported by the data in Table 5, which indicates minimal differences between the converted scale values and the force

sensor values. The close agreement between these values underscores the sensor's reliability and precision in detecting force.

Additionally, the mean relative error of 0.0187 demonstrates that the discrepancies between the measured values are very small. This low error percentage indicates that the force sensor's readings are consistently within 1.87% of the standard values, which is crucial for applications requiring precise force measurements. The relative error for each measurement is calculated using the formula (Equation 2):

$$\text{Relative Error} = \frac{|\text{Measured Value} - \text{True Value}|}{\text{True Value}} \tag{2}$$

Overall, the data validates the proportional relationship of the force sensor, confirming that it provides a dependable representation of the actual forces applied. This is evidenced by both the graphical (Figure 15) and tabular (Table 5) results.

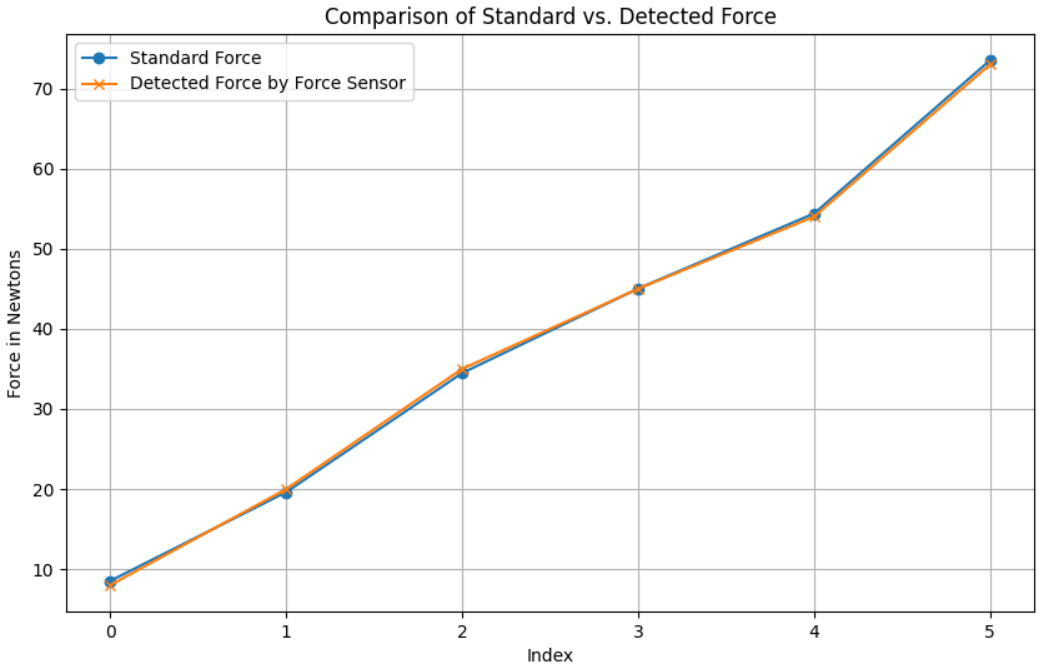


Figure 15: Comparison of Standard vs. Detected Force

Converted Scale Values (N)	Force Sensor Values (N)	Relative Error
8.53	8	0.06265012
19.62	20	0.01936799
34.46	35	0.01501792
45.02	45	0.00061962
54.37	54	0.00728816
73.49	73	0.00715326

Table 5: Comparison of Converted Scale Values and Force Sensor Values (in Newtons)

3.1.3 Conclusion

The verification process for the force sensor demonstrated that it provides accurate and reliable measurements, which are critical for the data acquisition in this project. Through validating both the linear relationship between the output voltage and the applied force and the accuracy of the proportional relationship, we confirmed that the force sensor performs consistently under varying conditions.

The successful validation process underscores the importance of the force sensor in our project. By ensuring that the force measurements are both accurate and proportional to the applied force, we can confidently use the data collected for precise analysis and decision-making. This reliability is essential for applications where accurate force measurement is crucial.

3.2 Bluetooth Module

The Bluetooth module in our micro-penetrometer system is critical for reliable communication between the host computer and the device. Ensures that data integrity and command execution are maintained throughout operations. The primary function of the Bluetooth module is to facilitate the transfer of control commands from the host to the device and to relay data, including force measurements and positional information, back to the host. Table 10 shows the requirement and verification table for Bluetooth module.

3.2.1 Verification Process

The verification of the Bluetooth module involved several key tests to confirm its operational integrity and reliability:

1. **Command Transmission:** Commands such as ”start,” ”return,” and horizontal movement instructions (e.g., ”move 1 1”) were sent from the host computer to the device. These commands were encoded in UTF-8 format to ensure compatibility and prevent data corruption.

2. **Motor Response Testing:** Upon receiving commands, the motor's response was observed to verify that it moved as directed. This test confirmed the system's ability to interpret and execute commands accurately.
3. **Data Integrity Check:** To test the integrity of the data received by the host, the probe was moved 50 cm without allowing the force sensor to contact any surface. During this test, data regarding the depth and force were recorded and plotted to assess any potential data loss or corruption.

3.2.2 Results

The results from the verification tests demonstrated that the Bluetooth module performed reliably under test conditions. The motor responded accurately to the received commands, and the integrity of the data transmitted back to the host was maintained. The plot of depth versus force showed consistent and accurate data points, confirming that the Bluetooth module supports our system's requirements for precise data handling and robust wireless communication.

3.3 Motors

The micro-penetrometer system utilizes multiple motors to control both horizontal and vertical movements with high precision. Verification of these motors is essential to ensure precise movement and positioning of the penetrometer's probe according to system commands. Table 11 shows the requirement and verification table for motors.

3.3.1 Horizontal Movement Verification

The horizontal movement is controlled by two motors, which navigate the probe across a 7x7 grid.

Procedure

1. **Command Execution:** Send sequential movement commands from (0,1) to (6,6) to the motors through the control program, directing them to move the probe across all grid points.
2. **Return Test:** After completing the grid movements, send a command (0,0) to ensure the motors return the probe to its original position.
3. **Accuracy Check:** By measuring the diameter of the pulley, we can calculate the theoretical distance traveled by the slider for each 360-degree rotation of the motor. After measuring the distance traveled by the slider multiple times, calculating the average value allows us to determine the error in the horizontal movement of the motor rotation.

Results The diameter of the pulley is $d = 19mm$. So for one cycle, the motor is expected to move $D = \pi * d = 59mm$. By measuring the distance traveled by the horizontal motor for multiple rotations and obtained their average. By utilizing the following equation 3, we determined the rotational error of the horizontal motor.

$$error = \frac{\frac{\Sigma Movement}{n} - D}{D} * 100\% = 0.85\% \tag{3}$$

3.3.2 Vertical Movement Verification

The vertical motor is responsible for controlling the depth of penetration by the probe, essential for collecting accurate subsurface data.

Procedure

1. **Depth Command Execution:** Issue commands to the vertical motor to rotate the probe down by 4 complete cycles (1440 degrees).
2. **Distance Measurement:** Employ an external measuring device to verify the actual distance the probe has traveled. Repeat this measure several times to ensure repeatability and reliability.
3. **Angle Verification:** Monitor and record the actual angles achieved by the motor using the control algorithm, checking for precision in achieving the required angular displacement.

Results By measuring the diameter of the lead screw, we determined that the slider should descend 20mm when the motor rotates 1440 degrees. We measured the distance traveled by the slider multiple times. When the motor rotated 1440 degrees, the slider descended 20.01mm; when it rotated 2880 degrees, the slider descended 40.03mm, and when it rotated 4320 degrees, the slider descended 60.06mm. From this data, we calculated the rotational error of the vertical motor by equation 4.

$$error = \frac{\frac{D_1+D_2+D_3}{3} - D_{expect]}}{D_{expect}} * 100\% = 0.1\% \tag{4}$$

3.4 Overall System Verification

The comprehensive verification process of the micro-penetrometer system's functionality included testing with four specific types of soil to ensure accuracy and reliability in diverse conditions. Table 12 shows the requirement and verification table for overall system requirements.

3.4.1 Soil Test Setup

Figure 16 shows layered soil sample setup used for the penetration test. The system was evaluated using the following soil samples, each with distinct physical properties:

- **Paving Stone:** Placed at the top layer, offering a hard surface.
- **Pastoral Soil:** Rich in organic matter, commonly used in grasslands and agricultural fields.
- **River Sandriversand:** Known for its gritty texture, typically found in riverbeds.
- **Coconut Soilcoconutsoil:** Made from coconut husk fibers, highly absorbent and lightweight.

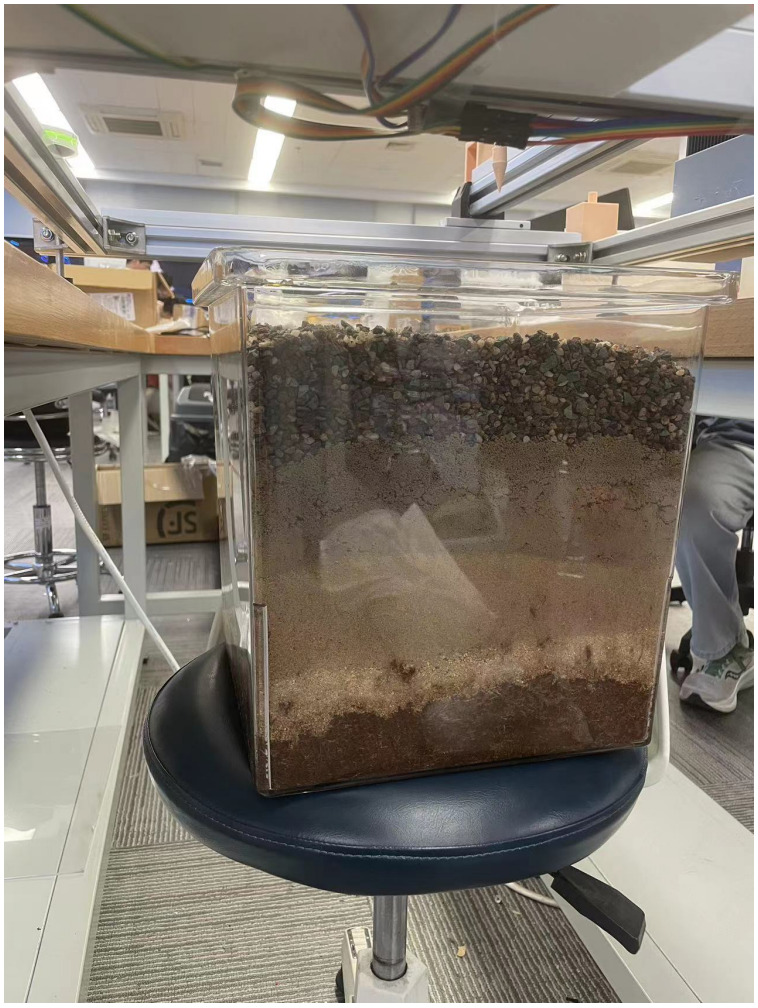


Figure 16: Layered soil sample setup used for the penetration test.

3.4.2 Procedure

The procedure involved allowing the probe to penetrate through the soil layers. If the force exceeded 90 N, the probe would return, and the data would not be recorded.

3.4.3 Results and Analysis

Figure 17 shows the force (N) over depth for position (3, 4). The graph illustrates the force required to penetrate each soil layer. The slope of the graph correlates with the properties of the different soil types:

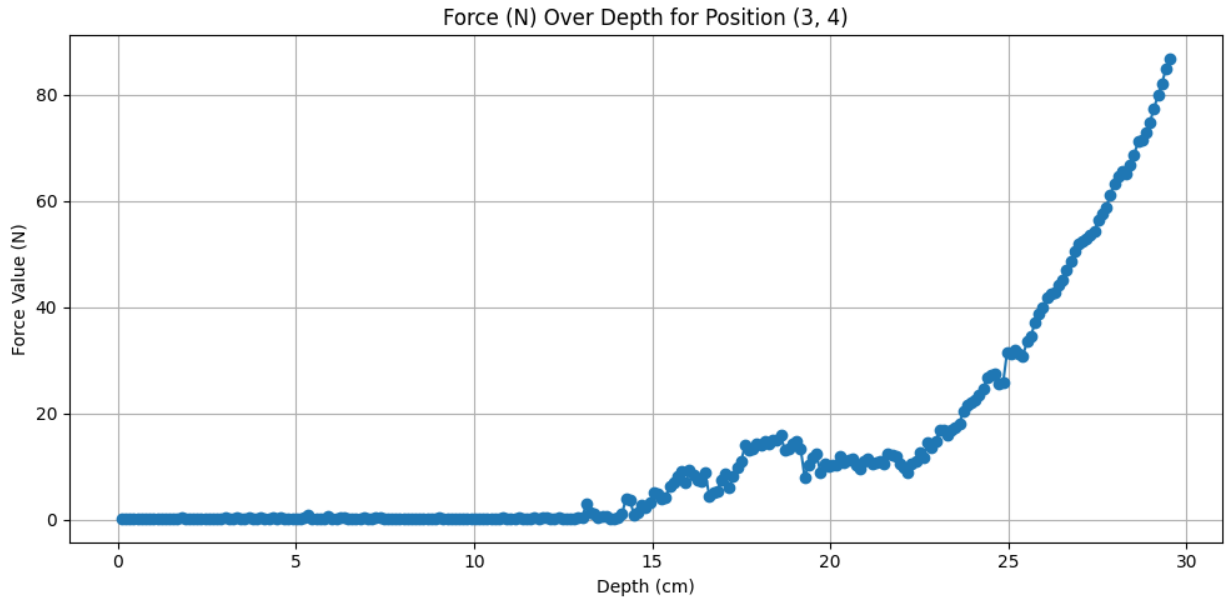


Figure 17: Force (N) versus depth (cm) for position (3, 4) showing the resistance encountered by the probe in different soil layers.

- From 0 to 13 cm, the force data is meaningless as the probe has not yet contacted the soil.
- From 13 to 20 cm, the force remains relatively low, corresponding to the macadam layer. This is because the macadam has larger pores and is located at the uppermost layer, resulting in lower pressure on the drill bit, allowing it to penetrate this layer easily.
- Beyond 20 cm, the smaller soil particles and their lower layer position result in a more compact structure. As a result, the pressure on the drill bit increases significantly with depth. The obtained data matches the images presented in the referenced paper, demonstrating that our device accurately reflects the expected data.

4 Costs

4.1 Labor

Name[1]	**Hourly Rate**[1]	**Hours**[1]	**Total**[1]	**Total x2.5**[1]
Chenghao Mo[1]	¥30[1]	200[1]	¥6000[1]	¥15000[1]
Xing Shen[1]	¥30[1]	200[1]	¥6000[1]	¥15000[1]
Zheyan Wu[1]	¥30[1]	200[1]	¥6000[1]	¥15000[1]
Chenxian Meng[1]	¥30[1]	200[1]	¥6000[1]	¥15000[1]
Total[1]			¥60000[1]	

Table 6: Labor Costs[1]

4.2 Parts

Table 7: Component Costs

Description	**Quantity**	**Manufacture**	**Vendor**	**Cost/unit**	**Total cost**
Li-Polymer Battery (14.8 V, 48 Wh)	1	Wei Zhen	Tao Bao	¥34.20	¥34.20
GGP Dual optical axis ball screw module guide rail with motor	1	Mei Ke Transmission	Tao Bao	¥310.00	¥310.00
57HB56L4/1.2Nm motor	2	Planetary Deceleration	Tao Bao	¥50.00	¥100.00
3030N1-Aluminum profile	4	YYDS	Tao Bao	¥24.00/m	¥56.00

Continued on next page

Table 7 – continued from previous page

Description	Quantity	Manufacturer	Vendor	Cost/unit	Total cost
3030ED-Aluminum profile	1	YYDS	Tao Bao	¥21.00/m	¥12.00
Manufacture service for Aluminum profile	4	YYDS	Tao Bao	¥1.00	¥4.00
3030 Corner code	10	YYDS	Tao Bao	¥1.20	¥12.00
Delivery fee for Aluminum profile	1	YYDS	Tao Bao	¥12.00	¥12.00
DYM-106 Micro pressure sensor	2	Ocean Sensor	Tao Bao	¥234.00	¥234.00
Bluetooth module	1	Zheng Dian Atom	Tao Bao	¥56.36	¥56.36
Stepper motor driver	3	Zheng Dian Atom	Tao Bao	¥114.86	¥344.58
Motor control development board	1	Zheng Dian Atom	Tao Bao	¥532.38	¥532.38
Plum blossom handle screw	4	Wu Xi Quality	Tao Bao	¥0.74	¥2.80
Large head screw	1	Guwan Ji	Tao Bao	¥7.82	¥7.82
Guide rail slide	3	Zhejiang Zhenhao	Tao Bao	¥30.00	¥90.00
Guide rail slide with lock	3	Zhejiang Zhenhao	Tao Bao	¥48.00	¥144.00
Belt pulley	4	AET Hardware	Tao Bao	¥9.00	¥36.00

Continued on next page

Table 7 – continued from previous page

Description	Quantity	Manufacturer	Vendor	Cost/unit	Total cost
Belt	2	Libose	Tao Bao	¥9.00	¥18.00
Load sensor transmitter	1	DaYang Sensor	Tao Bao	¥234.00	¥234.00
Garden soil	1	Plant and Garden	Tao Bao	¥18.62	¥18.62
River soil and sand	1	Plant and Garden	Tao Bao	¥26.68	¥26.68
Succulent pellet nutrient soil	1	Plant and Garden	Tao Bao	¥9.80	¥9.80
Coconut coir brick nutrient soil	1	Plant and Garden	Tao Bao	¥12.56	¥12.56
Box	1	Hao Hao Furniture	Tao Bao	¥43.80	¥43.80
Large capacity lithium battery	1	XinDun energy lithium battery	Tao Bao	¥460.00	¥460.00
Gradienter	1	Zhongke Long	Tao Bao	¥12.30	¥12.30
Screw foot pad	4	Niuge	Tao Bao	¥3.70	¥20.80
Hexagon socket head cap screw	100	Niuge	Tao Bao	¥4.60	¥4.60
Total					**¥3083.3**

4.3 Grand Total

Section	Total
Labor	¥60000
Parts	¥3083.3
Grand Total	¥63083.3

Table 8: Grand Total Costs (Labor + Parts)

5 Conclusions

The project has successfully led to the construction of an advanced micro-penetrometer system designed for detailed analysis of soil and snow structures. The system integrates three stepper motors: two for precise horizontal movement, and a third motor connected to a screw for controlling the vertical movement of the probe. This ensures accurate and controlled penetration into diverse materials.

Central to our system's functionality is the STM32 microcontroller, which serves as the command hub. It receives commands from a remote host computer through a Bluetooth module, dispatches these commands to the motors, and handles data from the force sensor. This data is critical for analyzing the structural properties of the material being penetrated and is sent back to the host for real-time visualization and analysis.

5.1 Accomplishments

The development of the micro-penetrometer system has led to several significant technological advancements and functional capabilities, enhancing its application in soil and snow analysis. Key accomplishments of this project are outlined below:

- **Precision Control:** The system uses Pulse Width Modulation (PWM) to control the motor's angle precisely, enabling meticulous positioning of the probe. Integration of multiple stepper motors allows for exact control over the probe's positioning and depth, ensuring precise measurements critical for reliable analysis.
- **Real-time Data Processing:** Data from the force sensor are transmitted in real time to a remote host via Bluetooth, and stored in a SQLite database. This setup removes storage limitations on the STM32 microcontroller and facilitates advanced data analysis, enhancing the system's efficiency and analytical capabilities.
- **User Interface Development:** A custom-developed control program on the host computer enhances user interaction, enabling precise device manipulation and real-time data visualization. This intuitive interface simplifies device operation and allows for immediate interpretation of collected data, crucial for field decision-making.

5.2 Uncertainties and Alternatives

Although the system fulfills most of the design specifications, there are areas requiring further refinement to fully optimize its performance and durability. Specifically, battery life during continuous operation and the system's reliability under extreme environmental conditions have been identified as critical aspects needing additional exploration.

- **Power Management:** To extend the operating life, enhancements to the power management system are necessary. This could involve integrating more efficient battery technologies or developing advanced energy management protocols that reduce power consumption during idle periods.

- **Material Adaptation:** The materials currently used need to be evaluated and possibly upgraded to enhance durability and performance under various environmental conditions, including extreme weather such as rain and snow. This adaptation may involve using weather-resistant materials or additional sealing processes.
- **Circuit Layout and Packaging:** There is a need to improve the circuit layout of each subsystem to maximize space efficiency and enhance the robustness of connections. Better packaging of components can contribute significantly to the system's portability and durability, making it more capable of handling adverse conditions.
- **Equipment Modularity and Disassembly:** Enhancing the design to allow easier disassembly and reassembly of the device will facilitate field repairs and maintenance. This modularity will also support the customization of the system based on specific field conditions or user requirements.

These improvements are essential for ensuring that the micro-penetrometer can operate reliably in a broader range of environmental scenarios and for prolonging the lifespan of the device under typical field usage conditions.

5.3 Ethical Considerations

Our project's commitment to ethical engineering practices is rooted in the principles outlined by the IEEE Code of Ethics[6]. We integrate these principles within the scope of our micro-penetrometer development:

1. **Public Safety**: We prioritize the safety and well-being of the public by ensuring the reliable and safe operation of our penetrometer, conducting extensive tests to mitigate potential hazards.
2. **Honesty**: We adhere to honesty in reporting findings and claims based on data obtained by our penetrometer, ensuring accuracy and verifiability.
3. **Technological Understanding**: We aim to enhance technological understanding and its appropriate application in soil and snow analysis, with implications for agriculture and disaster management.
4. **Technical Competency**: We maintain technical competence, embarking on tasks within our expertise, and disclose any limitations in knowledge or abilities.
5. **Criticism and Credit**: We welcome criticism constructively, acknowledge errors, and appropriately credit the contributions and efforts of others involved in the project.
6. **Avoiding Harm**: We are committed to avoiding harm, by designing a safe-to-handle device and securely enclosing all electronic components to prevent misuse or accidents.
7. **Professional Support**: We support our peers in professional development and uphold the ethical standards set forth by IEEE, fostering a culture of ethical engineering.

Each principle is actively practiced and reflected in our project methodology, ensuring an alignment of our technological advancements with ethical engineering standards.

5.4 Broader Impacts

The micro-penetrometer system is designed with the future in mind, capable of integration with additional sensors such as temperature and humidity detectors. This enhancement would enable a more comprehensive analysis of soil and snow properties, further expanding its applicability across various fields.

Globally, the micro-penetrometer has significant potential to impact agricultural productivity, archaeological research, and environmental monitoring. Economically, it offers a cost-effective solution for on-site soil and snow analysis, potentially reducing the need for extensive laboratory equipment. Environmentally, the device aids in the assessment of soil health and snow stability, contributing to sustainable land management and disaster prevention efforts.

In conclusion, the micro-penetrometer system stands as a testament to innovative engineering, combining precise mechanical design with sophisticated software integration to serve a variety of global needs in soil and snow analysis.

References

[1] D. E. Rolston, M. N. A. Bedaiwy, and D. T. Louie, "Micropenetrometer for in situ measurement of soil surface strength," *Soil Science Society of America Journal*, vol. 55, no. 2, pp. 481–485, 1991. DOI: 10.2136/sssaj1991.03615995005500020031x.

[2] M. Schneebeli and J. B. Johnson, "A constant-speed penetrometer for high-resolution snow stratigraphy," *Annals of Glaciology*, vol. 26, pp. 107–111, 1998.

[3] OpenEDV, *Documentation for the ATK-2MD4850 Module*, http://www.openedv.com/docs/modules/other/ATK-2MD4850.html, [Online; accessed 28-September-2023], 2023.

[4] OpenEDV, *ATK-DMF407 Development Board Documentation*, http://www.openedv.com/docs/boards/stm32dj/ATK-DMF407.html, [Online; accessed 28-September-2023], 2023.

[5] OpenEDV, *ATK-BLE02 IoT Module Documentation*, http://www.openedv.com/docs/modules/iot/ATK-BLE02.html?highlight=%E8%93%9D%E7%89%99, [Online; accessed 28-September-2023], 2023.

[6] Institute of Electrical and Electronics Engineers, *IEEE Governance Documents*, Accessed: 2023-09-28, 2023. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html.

Appendix A Requirement and Verification Table

Requirement Description	**Verification Procedure**
The force sensor must accurately measure forces with a linear response to applied loads.	1. Validate linearity by applying known weights (100g increments) and measuring output voltage. Record the voltage for each weight and ensure a linear trend in the graphed results. 2. Use a standard laboratory force gauge to apply a range of forces. Compare the sensor output to the gauge readings to verify linearity across the sensor's operational range.
The force sensor output voltage must remain stable under constant load conditions over a period of time.	1. Conduct a stability test by applying a fixed weight (e.g., 500g) and record the sensor's voltage output every 10 seconds for 5 minutes. 2. Analyze the recorded data for any fluctuations, and ensure that the variation is within the acceptable error margin (e.g., less than 1% variance).
The force sensor must have a calibrated zero point where no force results in zero voltage output.	1. Zero calibration test: Ensure that no weight is applied and adjust the sensor (if necessary) so the output reads zero volts. 2. Verify the zero point under various environmental conditions (e.g., changes in temperature and humidity) to ensure stability.

Table 9: Force Sensor Requirements and Verification

Requirement Description	**Verification Procedure**
Module must be discoverable by the host device for connection establishment.	Perform a scan on the host device and verify that the ATK-MW579 is listed among the discoverable devices.
Module must transmit data with integrity to the host device over an established Bluetooth connection.	1. Command transmission testing: Send commands such as ”start,” ”return,” and ”move 1 1” from the host. Ensure commands are received and executed correctly by the device. 2. Motor response and data integrity check: After sending commands, monitor motor response and verify correct movement. Simultaneously, check the integrity of data received at the host side. Conduct a no-contact test where the probe moves 50 cm, record and plot depth versus force data to ensure all data points are received accurately.

Table 10: Bluetooth Module Requirements and Verification

Requirement Description	**Verification Procedure**
Motors must ensure precise horizontal movement across a 7x7 grid without slippage or deviation.	1. Execute command sequence from (0,1) to (6,6) and verify each grid point is reached accurately using a position sensor. 2. Conduct a return test to original position (0,0) to check the accuracy of the reverse movement. 3. Measure and record the distance traveled for each motor cycle, compare with the theoretical distance ($D = \pi \times d$), and calculate the error percentage using equation 3.
Motors must accurately control the vertical movement of the penetrometer's probe to ensure precise depth penetration.	1. Send depth commands for multiple rotations (e.g., 1440, 2880, 4320 degrees) and use an external measuring device to verify the depth reached by the probe. 2. Repeat depth measurements to ensure consistency and reliability of vertical positioning. 3. Calculate the average descent per rotation, compare with expected descent (20mm per 1440 degrees), and compute the error as outlined in equation 4.

Table 11: Motor Requirements and Verification

Requirement Description	**Verification Procedure**
The system must accurately measure penetration depth and resistance in different soil types (River Sand, Pastoral Soil, Coconut Soil, Nutrient Soil).	1. Conduct individual soil testing by placing each soil type in a separate container. Use the probe to penetrate each sample while recording depth and resistance. 2. Ensure the system records consistent penetration depths and resistance values that align with the known properties of each soil type.
The system must effectively handle transitions between different soil layers and accurately reflect changes in resistance and depth.	1. Arrange the soils in layered sequences within a single container to mimic varied ground conditions. 2. As the probe penetrates through these layers, verify that the system accurately records shifts in resistance and depth corresponding to changes in soil types. 3. Review data to confirm the system's responsiveness and precision in detecting different soil layers.

Table 12: Overall System Requirements and Verification

Appendix B Motor, ADC and Bluetooth Communication

```
#include "./SYSTEM/sys/sys.h"
#include "./SYSTEM/delay/delay.h"
#include "./SYSTEM/usart/usart.h"
#include "./BSP/LED/led.h"
#include "./BSP/KEY/key.h"
#include "./BSP/LCD/lcd.h"
//#include "demo.h"
#include "./BSP/ATK_MW579/atk_mw579.h"
#include "./BSP/ADC/adc.h"
#include "./BSP/TIMER/stepper_tim.h"
#include "./BSP/STEPPER_MOTOR/stepper_motor.h"
#include <string.h>
#include <stdio.h>

#include "./BSP/RTC/rtc.h"
#include "./USMART/usmart.h"

#define DEMO_BLE_NAME           "ATK-MW579"
#define DEMO_BLE_HELLO          "HELLO ATK-MW579"
#define DEMO_BLE_ADPTIM         5


int id = 1;
extern uint8_t g_run_flag;
void show_mesg(void)
{
    lcd_show_string(10, 10, 220, 32, 32, "Group-12", RED);
    lcd_show_string(10, 47, 220, 24, 24, "MICRO-PENETROMETER", RED);
    //lcd_show_string(10, 76, 220, 16, 16, "ATOM@ALIENTEK", RED);

    printf("\n");
    printf("********************************\r\n");
    printf("STM32\r\n");
    printf("ATK-MW579\r\n");
    //printf("ATOM@ALIENTEK\r\n");
    printf("********************************\r\n");
    printf("\r\n");
}

void bluetooth(void)
{
    uint8_t ret;
    uint8_t key;
    uint8_t *recv_dat;

    uint16_t adcx;

    uint8_t start_hour, start_min, start_sec, start_ampm;
    uint8_t hour, min, sec, ampm;
    uint8_t year, month, date, week;
    uint8_t tbuf[40];
```

```
uint8_t send_flag=0;

float temp;
float voltage;
float force;


uint8_t flag = 0, t = 0;
uint8_t dir = 1;

uint8_t start_t;

int angle = 0;

char buf[32];

ret = atk_mw579_init(ATK_MW579_UART_BAUDRATE_115200);
if (ret != 0)
{
    printf("ATK-MW579 init failed!\r\n");
    lcd_show_string(30, 187, 200, 16, 16, "ATK-MW579 init failed!", BLUE);
    while (1)
    {
        LED0_TOGGLE();
        delay_ms(200);
    }
}

atk_mw579_enter_config_mode();
ret  = atk_mw579_set_name(DEMO_BLE_NAME);
ret += atk_mw579_set_hello(DEMO_BLE_HELLO);
ret += atk_mw579_set_tpl(ATK_MW579_TPL_P0DBM);
ret += atk_mw579_set_uart(ATK_MW579_UART_BAUDRATE_115200,
   ATK_MW579_UART_DATA_8, ATK_MW579_UART_PARI_NONE, ATK_MW579_UART_STOP_1
   );
ret += atk_mw579_set_adptim(DEMO_BLE_ADPTIM);
ret += atk_mw579_set_linkpassen(ATK_MW579_LINKPASSEN_OFF);
ret += atk_mw579_set_leden(ATK_MW579_LEDEN_ON);
ret += atk_mw579_set_slavesleepen(ATK_MW579_SLAVESLEEPEN_ON);
ret += atk_mw579_set_maxput(ATK_MW579_MAXPUT_OFF);
ret += atk_mw579_set_mode(ATK_MW579_MODE_S);
if (ret != 0)
{
    printf("ATK-MW579 config failed!\r\n");
    lcd_show_string(30, 187, 200, 16, 16, "ATK-MW579 config failed!", BLUE
       );
    while (1)
    {
        LED0_TOGGLE();
        delay_ms(200);
    }
}
```

```
printf("Connection Success\r\n");
lcd_show_string(30, 187, 200, 16, 16, "Connection Success", BLUE);

atk_mw579_uart_rx_restart();

while (1)
{

    rtc_get_time(&hour, &min, &sec, &ampm);
    rtc_get_date(&year, &month, &date, &week);
    sprintf((char *)tbuf, "Time:%02d:%02d:%02d", hour, min, sec);
    lcd_show_string(30, 170, 210, 16, 16, (char*)tbuf, RED);

    adcx = adc_get_result_average(ADC_ADCX_CHY, 10);

    lcd_show_xnum(120, 107, adcx, 5, 16, 0, BLUE);

    temp = (float)adcx * (3.3 / 4096);
    voltage = (float)adcx * (3.3 / 4096);
    adcx = temp;
    lcd_show_xnum(140, 127, adcx, 1, 16, 0, BLUE);

    temp -= adcx;
    temp *= 1000;
    force = voltage * 6 * 9.8;
    lcd_show_xnum(150, 127, temp, 3, 16, 0X80, BLUE);
    lcd_show_xnum(130, 147, force, 3, 16, 0X80, BLUE);


    key = key_scan(0);
    if(send_flag)
        atk_mw579_uart_printf("%f,%d\r\n",voltage, g_stepper.
           add_pulse_count);
    switch (key)
    {
        case KEY0_PRES:
        {
            atk_mw579_uart_printf("adc:%f\r\n",voltage);
            break;
        }
        case KEY1_PRES:
        {
            atk_mw579_wakeup_by_uart();
            break;
        }
        default:
        {
            break;
        }
    }

    recv_dat = atk_mw579_uart_rx_get_frame();
    if (recv_dat != NULL)
    {
```

```
            printf("%s\r\n", recv_dat);

            const char *start = "start";
            if (strncmp((const char*)recv_dat, start, strlen(start)) == 0) {
                flag = !flag;
                if(flag)
                {
                    g_stepper.add_pulse_count = 0;
                    if(g_run_flag == 0)
                    {
                        angle = 360 * 100;
                        id = 1;
                        g_stepper.angle = angle;
                        g_stepper.dir = CW;
                        angle = 0;
                        send_flag = 1;
                        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
                    }
                    flag = !flag;

                }
                else
                {
                    id = 1;
                    printf("sdhfksdjfksjd\r\n");
                    stepper_stop(id);
                }
            }


            const char *change = "change";
            if(strncmp((const char*)recv_dat, change, strlen(change)) == 0)
            {
                id = 1;
                dir = !dir;
                send_flag = !send_flag;
                stepper_star(id, dir);
//                stepper_pwmt_speed(set_speed+100,ATIM_TIMX_PWM_CH1);
                printf("hour:%d; min: %d; sec: %d\r\n",hour, min, sec);
                delay_ms(1000*((hour-start_hour)*60*60 + (min-start_min)*60 +
                    (sec - start_sec)));
                stepper_stop(id);
                dir = !dir;
            }

            const char *up = "up";
            if(strncmp((const char*)recv_dat, up, strlen(up)) == 0)
            {
                if(g_run_flag == 0)
                {
                    angle = 360 * 10;
                    id = 1;
                    g_stepper.angle = angle;
                    g_stepper.dir = CCW;
```

```
            angle = 0;
            stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
        }
    }

    const char *down = "down";
    if(strncmp((const char*)recv_dat, down, strlen(down)) == 0)
    {
        if(g_run_flag == 0)
        {
            angle = 360 * 10;
            id = 1;
            g_stepper.angle = angle;
            g_stepper.dir = CW;
            angle = 0;
            stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
        }
    }

    const char *re = "return";
    if(strncmp((const char*)recv_dat, re, strlen(re)) == 0)
    {
        g_run_flag = 0;
        angle = g_stepper.add_pulse_count*MAX_STEP_ANGLE;
        id = 1;
        send_flag = 0;
        g_stepper.angle = angle;
        g_stepper.dir = CCW;
        angle = 0;
        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
    }

    const char *move = "move";
    if(strncmp((const char*)recv_dat, move, 4) == 0)
    {
        int dx, dy;

        // Find the position of the first space in the message
        char *space = strchr((const char*)recv_dat, ' ');

        // Move the pointer to the beginning of the dx value
        space++; // Increment to skip the space

        // Extract dx and dy values from the message
        sscanf(space, "%d %d", &dx, &dy);

        // Print the values to confirm
        printf("dx = %d, dy = %d\n", dx, dy);

        if(dx < 0)
        {
            if(g_run_flag == 0)
            {
                angle = 360 * (-dx);
```

```
                        id = 3;
                        g_stepper.angle = angle;
                        g_stepper.dir = CW;
                        angle = 0;
                        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
                    }

                }
                else
                {
                    if(g_run_flag == 0)
                    {
                        angle = 360 * dx;
                        id = 3;
                        g_stepper.angle = angle;
                        g_stepper.dir = CCW;
                        angle = 0;
                        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
                    }
                }
                while(g_run_flag == 1)
                {
                    delay_ms(10);
                }
                if(dy < 0)
                {
                    printf("g_stepper.pulse_count = %d\n", g_stepper.
                       pulse_count);
                    printf("g_run_flag = %d\n", g_run_flag);
                    if(g_run_flag == 0)
                    {
                        angle = 360 * (-dy);
                        id = 2;
                        g_stepper.angle = angle;
                        g_stepper.dir = CW;
                        angle = 0;
                        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
//                          printf("g_stepper.pulse_count = %d\n", g_stepper.
   pulse_count);
//                          printf("g_run_flag = %d\n", g_run_flag);
                    }
                }
                else
                {
                    printf("g_stepper.pulse_count = %d\n", g_stepper.
                       pulse_count);
                    printf("g_run_flag = %d\n", g_run_flag);
                    if(g_run_flag == 0)
                    {
                        angle = 360 * dy;
                        id = 2;
                        g_stepper.angle = angle;
                        g_stepper.dir = CCW;
                        angle = 0;
```

```
                        stepper_set_angle(g_stepper.angle, g_stepper.dir, id);
//                        printf("g_stepper.pulse_count = %d\n", g_stepper.
    pulse_count);
//                        printf("g_run_flag = %d\n", g_run_flag);
                    }
                }

            }



            atk_mw579_uart_rx_restart();
        }

        delay_ms(300);
        if ((t % 20) == 0)
        {
            LED0_TOGGLE();  /*  200ms ,        LED0         */
        }
    }
}

int main(void)
{

    HAL_Init();
    sys_stm32_clock_init(336, 8, 2, 7);
    delay_init(168);
    usart_init(115200);
    usmart_dev.init(84);
    led_init();
    key_init();
    lcd_init();
    show_mesg();


    stepper_init(1000 - 1, 168 - 1);







    rtc_init();
    rtc_set_wakeup(RTC_WAKEUPCLOCK_CK_SPRE_16BITS, 0);



    adc_init();
    lcd_show_string(30, 87, 200, 16, 16, "ADC TEST", RED);
    lcd_show_string(30, 107, 200, 16, 16, "ADC_CH3_VAL:", BLUE);
    lcd_show_string(30, 127, 200, 16, 16, "ADC_CH3_VOL:", BLUE);
    lcd_show_string(147, 127, 200, 16, 16, ".", BLUE);
    lcd_show_string(30, 147, 200, 16, 16, "ADC_FORCE:", BLUE);
```

```
    lcd_show_string(30, 187, 200, 16, 16, "Bluetooth Status", BLUE);


    bluetooth();


}
```

Appendix C Control Program

```
import asyncio
import tkinter as tk
from tkinter import messagebox, scrolledtext
from bleak import BleakScanner, BleakClient
from threading import Thread
import sqlite3
from datetime import datetime
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

class BleakApp:
    def __init__(self, master, loop):
        self.master = master
        self.loop = loop
        master.title("Bluetooth Communication with Device")
        self.database_setup()
        self.setup_connection_page()
        self.last_position = None
        self.setup_close_event()  # Setup close event binding

    def setup_close_event(self):
        self.master.protocol("WM_DELETE_WINDOW", self.on_close)

    def on_close(self):
        # Run the asynchronous return_to_home task
        asyncio.run(self.return_to_home())
        self.master.destroy()

    async def return_to_home(self):
        if self.last_position is None:
            dx, dy = 0, 0  # Sending absolute position for the first click
        else:
            dx, dy = 0 - self.last_position[0], 0 - self.last_position[1]

        home_command = f"move {dx} {dy}"
        data_to_send = home_command.encode('utf-8')
        write_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50200406e"  # Replace with
           your characteristic UUID
        try:
            await self.client.write_gatt_char(write_uuid, data_to_send,
               response=False)
            print("Device returning to home position (0,0)")
        except Exception as e:
            print(f"Failed to send home command: {str(e)}")



    def database_setup(self):
        # Connect to the SQLite database (the database file will be created if
            it does not exist)
        self.conn = sqlite3.connect('adc_data.db')
```

```
        self.cursor = self.conn.cursor()
        # Create a table to store ADC values with a timestamp
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS adc_values (
                timestamp TEXT NOT NULL,
                x INTEGER NOT NULL,
                y INTEGER NOT NULL,
                adc_value REAL NOT NULL,
                angle REAL NOT NULL
            )
        ''')
        self.conn.commit()


    def setup_connection_page(self):
        self.connection_frame = tk.Frame(self.master)
        self.connection_frame.pack(padx=10, pady=10)

        tk.Label(self.connection_frame, text="Select a Bluetooth Device:").
           pack(pady=(0, 10))

        self.scan_button = tk.Button(self.connection_frame, text="Scan",
           command=self.initiate_scan)
        self.scan_button.pack()

        self.devices_listbox = tk.Listbox(self.connection_frame, width=50,
           height=10)
        self.devices_listbox.pack(pady=10)

        self.connect_button = tk.Button(self.connection_frame, text="Connect",
            command=self.connect_device)
        self.connect_button.pack(pady=(10, 0))

    def initiate_scan(self):
        asyncio.run_coroutine_threadsafe(self.scan_devices(), self.loop)

    async def scan_devices(self):
        self.devices_listbox.delete(0, tk.END)
        devices = await BleakScanner.discover()
        for device in devices:
            self.devices_listbox.insert(tk.END, f"{device.name} | {device.
               address}")

    def connect_device(self):
        selection = self.devices_listbox.curselection()
        if not selection:
            messagebox.showerror("Error", "Please select a device from the
               list.")
            return

        selected_device = self.devices_listbox.get(selection[0])
        address = selected_device.split('|')[1].strip()
        asyncio.run_coroutine_threadsafe(self.start_client(address), self.loop
           )
```

```
async def start_client(self, address):
    client = BleakClient(address)
    try:
        await client.connect()
        if client.is_connected:
            self.client = client
            self.setup_communication_interface()
            self.setup_grid_selection()
            self.setup_results_display()
            self.setup_overview_button()
            await self.manage_device_communication()
        else:
            messagebox.showinfo("Connection Status", "Failed to connect to
                the device.")
    except Exception as e:
        messagebox.showerror("Connection Error", str(e))

def setup_communication_interface(self):
    self.connection_frame.pack_forget()
    self.communication_frame = tk.Frame(self.master)
    self.communication_frame.pack(padx=10, pady=10)

    self.text_area = scrolledtext.ScrolledText(self.communication_frame,
       width=60, height=10)
    self.text_area.pack(pady=10)

    self.entry = tk.Entry(self.communication_frame, width=40)
    self.entry.pack(side=tk.LEFT, padx=(10, 0))

    self.send_button = tk.Button(self.communication_frame, text="Send",
       command=self.send_message)
    self.send_button.pack(side=tk.LEFT, padx=10)

    # self.up_button = tk.Button(self.communication_frame, text="Up",
       command=lambda: self.send_predefined_message("up"))
    # self.up_button.pack(side=tk.LEFT, padx=10)

    # self.down_button = tk.Button(self.communication_frame, text="Down",
       command=lambda: self.send_predefined_message("down"))
    # self.down_button.pack(side=tk.LEFT, padx=10)

    self.stop_button = tk.Button(self.communication_frame, text="Return",
       command=lambda: self.send_predefined_message("return"))
    self.stop_button.pack(side=tk.LEFT, padx=10)

    self.stop_button = tk.Button(self.communication_frame, text="Start",
       command=lambda: self.send_predefined_message("start"))
    self.stop_button.pack(side=tk.LEFT, padx=10)

def send_predefined_message(self, message):
    self.append_text(f"Sent: {message}")
    data_to_send = message.encode('utf-8')
    write_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50200406e"
```

```
        # Send the "power" message immediately.
        asyncio.run_coroutine_threadsafe(
            self.client.write_gatt_char(write_uuid, data_to_send, response=
                False),
            self.loop
        )

        # If the message is "power", schedule to send "stop" after 20 seconds.
        # if message == "power":
        #     self.master.after(32500, self.send_stop_message)

    def send_stop_message(self):
        stop_message = "stop"
        self.append_text(f"Sent: {stop_message}")
        data_to_send = stop_message.encode('utf-8')
        write_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50200406e"

        # Send the "stop" message after a delay.
        asyncio.run_coroutine_threadsafe(
            self.client.write_gatt_char(write_uuid, data_to_send, response=
                False),
            self.loop
        )

    def setup_grid_selection(self):
        self.grid_frame = tk.Frame(self.master)
        self.grid_frame.pack(side=tk.TOP, padx=10, pady=10)

        # Description label with pack
        tk.Label(self.grid_frame, text="Select a cell to send the probe to the
            corresponding position:", font=('Helvetica', 10)).pack(pady=(0,
            10))

        # Sub-frame for grid, using a different frame to use grid layout
        self.button_grid_frame = tk.Frame(self.grid_frame)
        self.button_grid_frame.pack(pady=(10, 10))  # Using pack for the frame
            that contains the grid

        # Creating a grid of buttons inside the sub-frame
        self.grid_buttons = {}
        self.selected_buttons = set()
        for i in range(7):
            for j in range(7):
                button = tk.Button(self.button_grid_frame, text=f'{i},{j}',
                    width=6, height=2)
                button.grid(row=i, column=j, padx=2, pady=2)
                self.grid_buttons[(i, j)] = button
                button.config(command=lambda i=i, j=j, btn=button: self.
                    handle_grid_click(i, j, btn))

    def handle_grid_click(self, i, j, button):
        # Check if it's the first click
        if self.last_position is None:
```

```
            # For the first click, treat the clicked position as relative to
                (0,0)
            dx, dy = i, j  # Sending absolute position for the first click
        else:
            # For subsequent clicks, calculate relative motion
            dx, dy = i - self.last_position[0], j - self.last_position[1]

        # Update the last position to the current position
        self.last_position = (i, j)

        # Change the button color to indicate it has been selected at least
            once
        if button not in self.selected_buttons:
            button.config(bg='lightblue')  # Change color to light blue
            self.selected_buttons.add(button)

        # Schedule the database record insertion and motion command sending
        asyncio.run_coroutine_threadsafe(self.send_motion_to_device(dx, dy),
            self.loop)


async def send_motion_to_device(self, dx, dy):
    message = f'move {dx} {dy}'
    data_to_send = message.encode('utf-8')
    write_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50200406e"
    await self.client.write_gatt_char(write_uuid, data_to_send, response=
        False)
    self.append_text(f"Motion sent: {dx}, {dy}")


def setup_results_display(self):
    self.results_frame = tk.Frame(self.master)
    self.results_frame.pack(side=tk.BOTTOM, padx=10, pady=10, fill=tk.X,
        expand=True)

    self.results_label = tk.Label(self.results_frame, text="Results (Time
        - ADC Value for each (x, y))")
    self.results_label.pack()

    self.results_text = scrolledtext.ScrolledText(self.results_frame,
        height=10)
    self.results_text.pack(fill=tk.BOTH, expand=True)


async def notification_handler(self, sender, data):
    data_str = data.decode('utf-8').strip()
    # print(f"Received data: {data_str}")

    try:
        # Split data based on comma and remove any surrounding whitespace
        parts = [part.strip() for part in data_str.split(',')]
        if len(parts) != 2:
            raise ValueError(f"Expected 2 parts but got {len(parts)}: {
                parts}")
```

```
            adc_value, angle_value = map(float, parts)  # Convert both parts
                to float
            # print(adc_value, angle_value)

            # Add milliseconds to the timestamp
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]

            # Call the database insert function in an executor to avoid
                blocking the event loop
            await self.loop.run_in_executor(None, self.insert_db_record,
                timestamp, adc_value, angle_value*0.225)
            # print(f"Processed ADC: {adc_value}, Angle: {angle_value}")

        except ValueError as e:
            print(f"Error processing notification: {e}")
            self.append_text(f"Error processing notification: {e}")

    def insert_db_record(self, timestamp, adc_value, angle_value):
        try:
            # Perform the SQLite operations
            conn = sqlite3.connect('adc_data.db')
            cursor = conn.cursor()
            cursor.execute('INSERT INTO adc_values (timestamp, x, y, adc_value
                , angle) VALUES (?, ?, ?, ?, ?)',
                           (timestamp, self.last_position[0], self.last_position
                               [1], adc_value, angle_value))
            conn.commit()
        except sqlite3.Error as e:
            # Log or handle the error
            print(f"Database error: {e}")
        finally:
            conn.close()

    async def manage_device_communication(self):
        notify_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50300406e"
        # Change 'notification_handler' to 'self.notification_handler'
        await self.client.start_notify(notify_uuid, self.notification_handler)
        self.append_text("Subscribed to notifications. Listening for messages
            from the device...")

    def append_text(self, text):
        # Ensure the UI is updated in a thread-safe way
        self.text_area.after(0, lambda: self.text_area.insert(tk.END, f"{text
            }\n"))
        self.text_area.after(0, lambda: self.text_area.see(tk.END))

    def setup_results_display(self):
        self.results_frame = tk.Frame(self.master)
        self.results_frame.pack(side=tk.BOTTOM, padx=10, pady=10, fill=tk.X,
            expand=True)

        # Variable to hold the currently selected (x, y) pair
        self.selected_xy = tk.StringVar(self.master)
```

```
        self.selected_xy.set('Select Coordinates')  # default value

        # Dropdown menu to select (x, y) pair
        self.xy_menu = tk.OptionMenu(self.results_frame, self.selected_xy, '
            Select Coordinates')
        self.xy_menu.pack(pady=10)

        # Attach an event to the dropdown to refresh the list whenever it's
            clicked
        self.xy_menu.bind('<Button-1>', self.update_dropdown)

        self.plot_button = tk.Button(self.results_frame, text="Plot ADC Values
            ", command=self.plot_adc_values)
        self.plot_button.pack()

    def update_dropdown(self, event=None):
        # Fetch unique (x, y) pairs from the database to populate the dropdown
        connection = sqlite3.connect('adc_data.db')
        cursor = connection.cursor()
        cursor.execute('SELECT DISTINCT x, y FROM adc_values ORDER BY x, y')
        coordinates = cursor.fetchall()
        connection.close()

        # Clear the current options in the OptionMenu
        menu = self.xy_menu["menu"]
        menu.delete(0, "end")
        for coordinate in coordinates:
            menu.add_command(label=coordinate, command=lambda value=coordinate
                : self.selected_xy.set(value))

        # If no data is available, set to default value
        if not coordinates:
            self.selected_xy.set('Select Coordinates')

    def plot_adc_values(self):
        selected_pair = self.selected_xy.get()
        if selected_pair == 'Select Coordinates':
            messagebox.showerror("Selection Error", "Please select valid
                coordinates.")
            return

        # Extract x, y from the selected pair string
        x, y = map(int, selected_pair.strip('()').split(','))

        # Fetch data from the database for the selected (x, y) pair
        connection = sqlite3.connect('adc_data.db')
        cursor = connection.cursor()
        # Fetching both adc_value and angle
        cursor.execute('SELECT timestamp, adc_value, angle FROM adc_values
            WHERE x=? AND y=? ORDER BY timestamp', (x, y))
        records = cursor.fetchall()
        connection.close()

        if not records:
```

```
            messagebox.showinfo("No Data", "No data available for this
                position.")
            return

        # Prepare data for plotting
        timestamps = [datetime.strptime(record[0], '%Y-%m-%d %H:%M:%S.%f') for
            record in records]
        adc_values = [record[1] * 120 for record in records]
        angles = [record[2] for record in records]

        # Calculate depth from angle
        depths = [(angle / 360) * 0.5 for angle in angles]

        # Plotting the data
        plt.figure(figsize=(10, 5))
        plt.plot(depths, adc_values, marker='o', linestyle='-')
        # plt.title(f'ADC Values Over Depth for Position ({x}, {y})')
        plt.title(f'Force Over Depth for Position ({x}, {y})')
        plt.xlabel('Depth (cm)')
        # plt.ylabel('ADC Value')
        plt.ylabel('Force Value')
        plt.grid(True)
        plt.tight_layout()
        plt.show()

    def setup_overview_button(self):
        # Adding a frame for the plot button to keep the layout organized
        self.plot_frame = tk.Frame(self.master)
        self.plot_frame.pack(padx=10, pady=10, fill=tk.X)

        # Button for plotting the overview of all data
        self.overview_plot_button = tk.Button(self.plot_frame, text="Plot Data
            Overview", command=self.plot_overview)
        self.overview_plot_button.pack()

    def plot_overview(self):
        # Connect to the database and fetch all data
        conn = sqlite3.connect('adc_data.db')
        cursor = conn.cursor()
        cursor.execute('SELECT x, y, adc_value, angle FROM adc_values')
        data = cursor.fetchall()
        conn.close()

        if not data:
            messagebox.showinfo("No Data", "No data available to plot.")
            return

        # Prepare data for plotting
        xs, ys, zs, colors = [], [], [], []
        for x, y, adc, angle in data:
            depth = (angle / 360) * 0.5  # Calculate depth based on angle
            xs.append(x)
            ys.append(y)
            zs.append(depth)
```

```
            colors.append(adc*120)  # Use ADC value as color scale

        # Create a 3D scatter plot
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')

        scatter = ax.scatter(xs, ys, zs, c=colors, cmap='viridis', marker='o',
            edgecolor='k', s=50, depthshade=True)
        # fig.colorbar(scatter, ax=ax, label='ADC Value')
        fig.colorbar(scatter, ax=ax, label='Force Value')

        ax.set_xlabel('X Position')
        ax.set_ylabel('Y Position')
        ax.set_zlabel('Depth (cm)')
        # plt.title('Overview of ADC Values Across Positions and Depths')
        plt.title('Overview of Force Values Across Positions and Depths')

        ax.invert_xaxis()
        plt.xticks(range(7))
        plt.yticks(range(7))

        plt.show()

    def send_message(self):
        message = self.entry.get()
        if message.lower() == 'exit':
            self.master.quit()
            return
        self.entry.delete(0, tk.END)
        data_to_send = message.encode('utf-8')
        write_uuid = "9ecadc24-0ee5-a9e0-93f3-a3b50200406e"
        asyncio.run_coroutine_threadsafe(
            self.client.write_gatt_char(write_uuid, data_to_send, response=
               False),
            self.loop
        )
        self.append_text(f"Sent: {message}")

    def append_text(self, text):
        self.text_area.insert(tk.END, f"{text}\n")
        self.text_area.see(tk.END)

def run_tkinter_loop(root):
    root.mainloop()

def main():
    root = tk.Tk()
    loop = asyncio.new_event_loop()
    app = BleakApp(root, loop)

    loop_thread = Thread(target=loop.run_forever)
    loop_thread.start()

    run_tkinter_loop(root)
```

```
    loop.call_soon_threadsafe(loop.stop)
    loop_thread.join()

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