Four-Axis Vacuum Stage for Advanced Nano-Manufacturing

ECE 445 Senior Design Report

Group 5

Songyuan Lyu

Yanjie Li

Xingjian Kang

Yanghonghui Chen

TA:

Boyang Shen

Supervisor:

Oleskiy Penkov

Abstract

As the development of nanotechnology, there is a requirement for nanocoating with higher precision. Currently, nanocoating has been applied in a variety of fields, such as surface engineering, aero-engineering and material science. The coatings are used to enhance the mechanical properties of the materials, reduce the friction of different surfaces, and provide some reagents for some enzyme reactions to increase the reaction efficiency. However, although nanocoating on a flat surface or a 2D frame has been deeply studied, there is a lack of research on nanocoating performed on irregular objects. This limits the progress and restricts the use of artificial joints and dental implants in biomedical industries. Thus, a four-axis vacuum stage for advanced nano-manufacturing has been designed and fabricated to realize nanocoating in 3D frame with high uniformity and quality. The vacuum stage is a four degrees of freedom (DOF) robotic arm made of aluminum. It is composed of four electrical motors, four reducers, a microcontroller, four motor controllers, a wireless control module and other aluminum structural components. The vacuum stage will be integrated into the nanocoating machine in Advanced Nanocoating Lab, and coating experiments and tribo-testings will be performed to prove the superiority of the vacuum stage.

Contents

1 Introduction

1.1 Current Nanocoating Techniques

Nanocoating, as a critical technique in nanotechnology, can be used to control the morphology of a material and achieve enhanced or multifunctional properties of the material [1]. It promotes progress in many different fields, such as surface engineering, aero-engineering, and material sciences. The working principle of nanocoating is to form a membrane that has a shape similar to the initial template. The nanocoating film is defined to have a thickness smaller than 100 nm, or the second phase nanoparticle is spread to the first phase matrix [1].
In industry, there are many advantages of nanocoating. For example, it can enhance the mechanical properties of some materials. These materials can be used to manufacture some structural components. In addition, the coating film can also increase the corrosion resistance of some materials. These materials can be used to produce some medical devices and increase the lifetime of these instruments [2] [3].
With the development of nanotechnology, a variety of nanocoating methods are studied to produce high-quality coatings. Some conventional nanocoating methods include spray coating and direct precipitation [4]. However, these coating methods may result in extra residual stresses and delamination. Thus, they will not retain strong mechanical stability. In comparison to these traditional nanocoating methods, the mainstream nanocoating technique is the physical vaporization deposition (PVD) method. One of the most popular PVD methods is magnetron sputtering. This method can achieve better coverage and adhesion of the coating film [5]. During the operation of magnetron sputtering, firstly, inert gas such as Argon will be input into a vacuum system. Then, a voltage will be applied to the electrodes, and the plasma will be formed. The inert gas will be ionized and be accelerated to sputter onto the cathode, which is composed of the target material. The target material will become versatile and be transported to deposit on the substrate, as shown in Fig. 1.

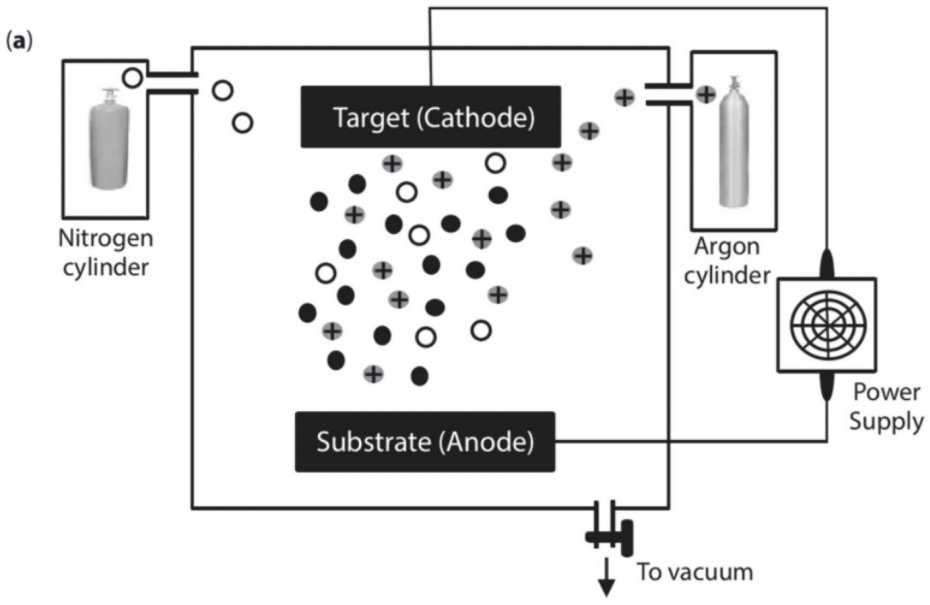


Figure 1: A schematic of magnetron sputtering process

The magnetron sputtering method allows the utilization of a small amount of materials to de-

posit the film. The film has enhanced mechanical properties and uniformity.
Integrating a multi-axis stage into the magnetron sputtering process is an innovative attempt. The vacuum stage should be able to operate normally in a high vacuum and high temperature environment, and it should not affect the operation of other steps during the coating process.

1.2 Economic Benefit & Demand

The nanofilm market represents a dynamic segment within the advanced materials industry, characterized by the production and application of ultra-thin films at the nanoscale. These films, typically measuring just a few nanometers in thickness, are designed to deliver unique properties such as enhanced barrier protection, optical clarity, and improved mechanical strength. As industries increasingly seek innovative solutions to their challenges, nanofilms have emerged as a critical technology across various applications, including electronics, packaging, and healthcare. This growing interest reflects broader trends toward miniaturization and efficiency in product design. Thus, the nanofilm market size reached 5.1 billion US dollar in 2023 and is projected to grow to 12.4 billion US dollar by 2030, with a compound annual growth rate (CAGR) of 12.1 Percent from 2024 to 2030. [6]

Table 1: Nanofilm Market Size and Growth Projections

Indicator	**Value**	**Notes**
2023 Market Size	US$5.1 billion	Base year data
2030 Projected Market Size	US$12.4 billion	Forecast (2024–2030 compound growth)
Compound Annual Growth Rate (CAGR)	12.1%	Period: 2024–2030

1.3 Motivation & Objective

Currently, most nanocoating methods are performed in a 2D frame (on a flat surface), specifically for a sample with regular shape. Although some popular nanocoating methods such as magnetron sputtering are also used to coat irregularly shaped objects, it takes a long time to perform the operation, and the coating film has low uniformity. It is a critical disadvantage when magnetron sputtering is used to perform nanocoating for some medical implants such as artificial joints. [7].
The objective is to design a structure to achieve magnetron sputtering in a three-dimensional frame in a vacuum environment. After investigation, implementing a robotic arm in the nanocoating machine can realize movement in a 3D frame with different postures [8]. Thus, the aim is to integrate a robotic arm into the nanocoating machine, and the robotic arm should satisfy the requirement to operate in a vacuum and high-temperature environment.

2 Design

2.1 Design procedure

2.1.1 Mechanical System

This section outlines the iterative design, key decisions, and engineering principles for the Four-Axis Vacuum Stage robotic arm, covering two main iterations that addressed early prototype deficiencies.

A. Joint 1 (Base Interface)

Function: Primary interface with the coating machine's central axis, providing stable base rotation. **Chosen Approach (V1 & V2):** Custom aluminum alloy assembly with integrated stepper motor and reducer. **Alternatives:** Direct motor mount with high-precision bearing; off-the-shelf vacuum rotary stage. **Justification:** Aluminum was selected for its vacuum compatibility and strength-to-weight ratio. A custom, CNC-machined design ensures precise integration with the coating machine and arm, optimized motor/reducer placement, and adequate torque, proving more cost-effective than specialized off-the-shelf stages for this application. Design relied on CAD (e.g., Fusion 360) and material selection. **Circuit Function:** `[Block Diagram: MCU -> Driver -> Stepper -> Reducer -> Joint1 Output]`
Function: Controller signals drive the motor via a driver; the reducer increases torque for arm base rotation.

B. Link 1 (J1 to J2 Connection)

Function: Transmits motion/forces; supports subsequent arm sections. **Chosen Approach (V1 & V2):** Standard 2020 aluminum extrusion. **Alternatives:** Custom CNC aluminum link; carbon fiber tube. **Justification:** 2020 extrusion offers modularity via T-slots, sufficient stiffness for the loads, and is highly cost-effective and available compared to custom CNC or carbon fiber. Design considered beam deflection ($\delta \propto PL^3/EI$, Eq. B.1) and bending stress ($\sigma = My/I$, Eq. B.2) using CAD and FEA.

C. Joint 2 (Shoulder Pitch)

Function: High-torque pitch motion for lifting the main payload. **V1 Design:** Aluminum frame, stepper motor with PLA gearing/direct drive, and a synchronous belt (which failed due to slippage and material degradation in vacuum/temperature). **V2 Chosen Approach:** Retained aluminum frame but upgraded actuation to a **screw motor (lead screw mechanism)**. **V2 Alternatives:** High-ratio metal gearbox; Harmonic Drive. **V2 Justification:** The screw motor provides high torque, stability, and self-locking capability (eliminating V1's slippage issues), and allows manual torque adjustment. It offered a better cost/performance balance than a harmonic drive. Design utilized lead screw mechanics ($F_{\text{axial}} \propto T_{\text{motor}}/p$; $T_{\text{joint}} = F_{\text{axial}} \cdot r_{\text{eff}}$) with CAD/FEA for the redesigned components. **Circuit Function (V2):** `[Block Diagram: MCU -> Driver -> Stepper (Lead Screw) -> Nut -> Linkage -> Joint2 Output]`
Function: Stepper rotates lead screw; linear nut motion converts to angular joint motion.

D. Joint 3 (Elbow Pitch)

Function: Pitch motion for the end-effector section. **Chosen Approach (V1 & V2):** Aluminum housing, actuated by a geared stepper motor, suitable for the lower end-effector mass. **Alternatives:** Direct drive motor; miniature screw drive. **Justification:** A geared stepper offers a good balance of torque, size, and cost for this joint. Torque calculations ($T_{\text{req}} \propto mgl \cdot \text{SF}$, Eq. D.1) confirmed adequacy. **Circuit Function:** `[Block Diagram: MCU -> Driver -> Geared Stepper -> Joint3 Output]`
Function: Controlled angular motion for the final arm segment.

E. Drivetrain System (Overall)

V1 Approach: PLA gearing and synchronous belts (failed due to slippage and material degradation). **V2 Chosen Approach:** Upgraded to robust metal gear-driven mechanisms (often integrated into steppers) and a specialized screw motor for Joint 2, enhancing torque, reliability, and environmental resistance.

F. Braking System (V2)

Function: Prevent unintended arm movement on power-off. **Chosen Approach (V2):** Motor-integrated electromagnetic braking system. **Alternatives:** Mechanical brakes; reliance on self-locking gearboxes. **Justification:** Electromagnetic brakes offer fail-safe operation (engage on power loss), provide an integrated/compact solution, allow controlled engagement, and enhance safety. **Circuit Function:** `[Block Diagram: Motor Power -> Control Signal -> Brake Coil]`
Function: Brake coil energized to release during operation; de-energizes to engage brake on power loss/hold.

2.1.2 Control System

In order to reach the control of the robotic arm to hold up the substrate to move the position and adjust the attitude to receive the proper coating, it is necessary to control at least four degrees of freedom. At first, one Arduino board connect with four motors is considered and tested.

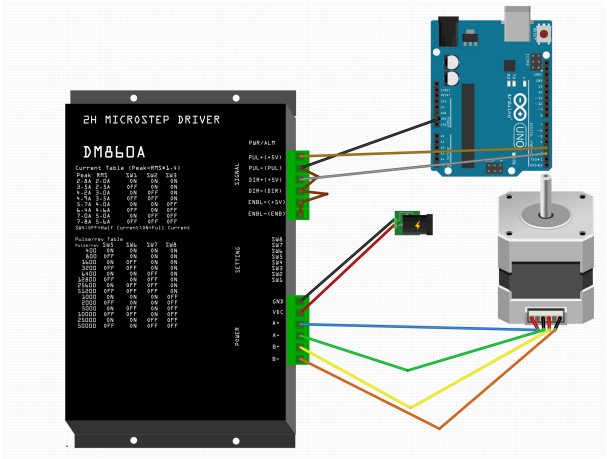


Figure 2: One Unit of Stepper Connection

Due to the lack of performance and protection, this design was eliminated. In the second iteration, many industrial-level features were considered:

1. RS485 Protocol: Widely used in industrial signal transmission, long transmission distance, strong anti-interference.

2. Optical Coupler: Optocouplers are characterized by mutual isolation between inputs and outputs, unidirectional transmission of electrical signals, and thus have good electrical insulation and anti-interference capability.

STM32 was chosen for its widely use and high stability to be the upper computer in the control system. The ”ZhengDianYuanZi STM32F407IG Industrial Control Development Board” was considered for it internal TTL to RS485 converter and built-in optical coupler. Normal stepper controller uses ENA, DIR, PUL to control. But a controller with built-in RS485 processing capability was prefered in this design. So ”ZDT Emm42 stepper controller” with optical coupler version was chosen.

2.1.3 PCB Design

As shown in Figure 4, close-up view of the pin interface of the EM42 motor driver and STM32 MCU board, critical connections with high lighting (e.g., power (A +, G) and RS485 A / B signals).

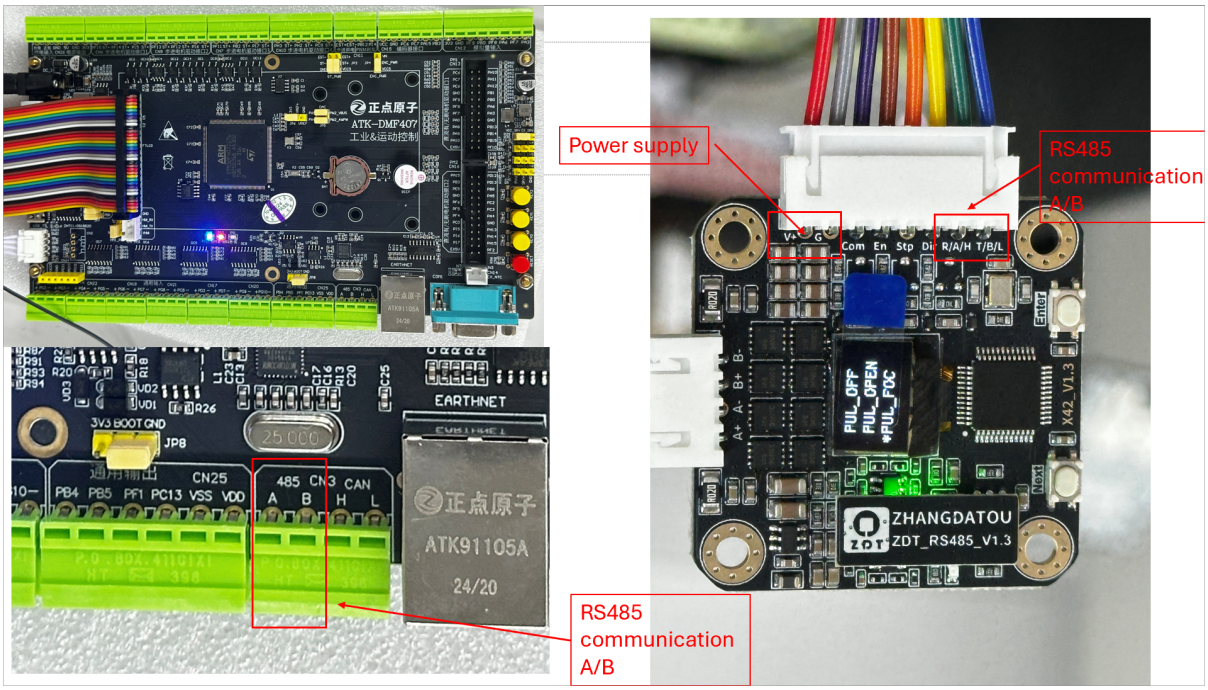


Figure 3: Close-up View of MCU and Motor Driver

At the top level, our PCB must serve two functions: distribute 24 V power to four EM42 motor drivers with minimal noise or voltage drop, and carry a robust, half-duplex RS-485 differential bus from the STM32 MCU to those same drivers. For each function we examined two architectures. For power we compared a centralized bus (one 4-way block feeding all drivers in parallel) versus individual point-to-point regulators on each motor-driver connector; we chose the former because the motors draw up to 2 A each only briefly and the single regulated 24 V

rail—with properly sized copper pours and decoupling—yields lower cost, smaller board area, and simpler thermal management. For communication we evaluated a linear "daisy-chain" bus topology against our star-style, four-port breakout with parallel termination; the star approach on PCB—with a single A/B entry followed by four equal-length, controlled-impedance traces—ensures equal signal delay and minimal stub reflections, and allows us to integrate both 120 Ω end-of-line termination resistors directly at the farthest connector blocks.

2.1.4 User Interface

Buttons: Figure 5 shows the physical buttons built into the STM32 MCU board. We plan to use the buttons to trigger a series of movement sets of the stepper motors.

Figure 4: Physical Buttons

TFT screen:

- Open-Loop vs. Closed-Loop Display

  Choice: We are using an open-loop motor control architecture, so we cannot rely on real-time feedback of actual motor states (position or velocity).

  Consequence: The screen must show the commanded values—set velocity, target position, and the state machine's phase—instead of actual readings.

  Workaround: We periodically poll the motor controller via RS-485 (every 20 main-loop iterations) and timestamp each poll so that, even though it's not truly synchronous feedback, we can estimate when motors should have reached their targets or detect errors via CRC flags.

- What to Display

  Global Status Bar: current state of the move-set state machine (IDLE, STEP1, ..., DONE).

  Per-Motor Panels: for each of the four motors, display

  Motor ID and whether it's "RUN" or "STOP."

  Commanded speed (RPM) and direction (CW/CCW).

  Target position (degrees).

  Message Area: transient text messages ("Running Main Program," "STOP," etc.) triggered by button presses or errors.

- Layout and Readability

  Fixed-width fields ensure that updates only overwrite old text, avoiding display artifacts.

  Color Coding:

  Blue for labels and normal info

Green for “RUN” status

Red for “STOP” or error conditions

Font Size and Positioning: all text at 16-pixel font, with consistent X/Y offsets so each line and panel aligns neatly.

- Timing and Refresh Strategy

  Non-blocking main loop: a 1 ms “HALDelay” ensures 1 kHz loop, so User Interface updates remain smooth without halting motor logic.

  Periodic Polling: every 20 iterations (20 ms), send read-back commands (EmmV5ReadSysParams) to each motor and then call “Translatereceiveddata()” once per poll to refresh the UI panels.

2.2 Actuator

Servos was considered for its simplicity and convenience. Servomotors was considered for high accuracy with feedback control. Stepper motors and reduction gears were considered for step-by-step control.
After evaluating the environment in the nano-coating machine, which is high temperature and high radiation. The Servo was negated for not enough accuracy .And the Servomotors have the built-in control, which is likely to be destroyed in the machine. Stepper motor system is further considered to combine with reduction gear and screw rod.

2.3 Design details

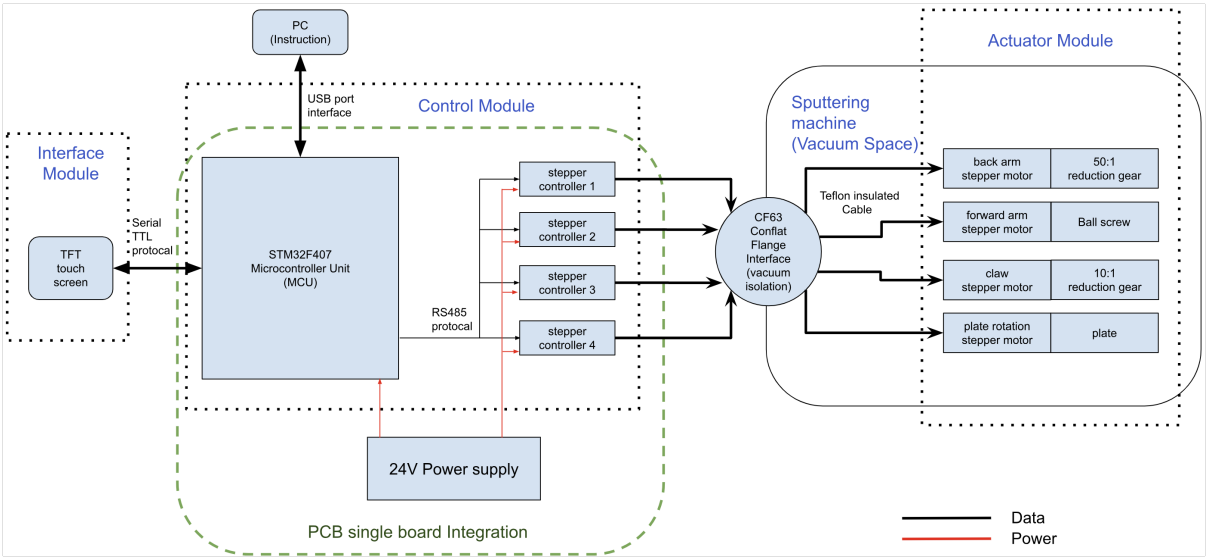


Figure 5: Block diagram of the system

2.3.1 Control System

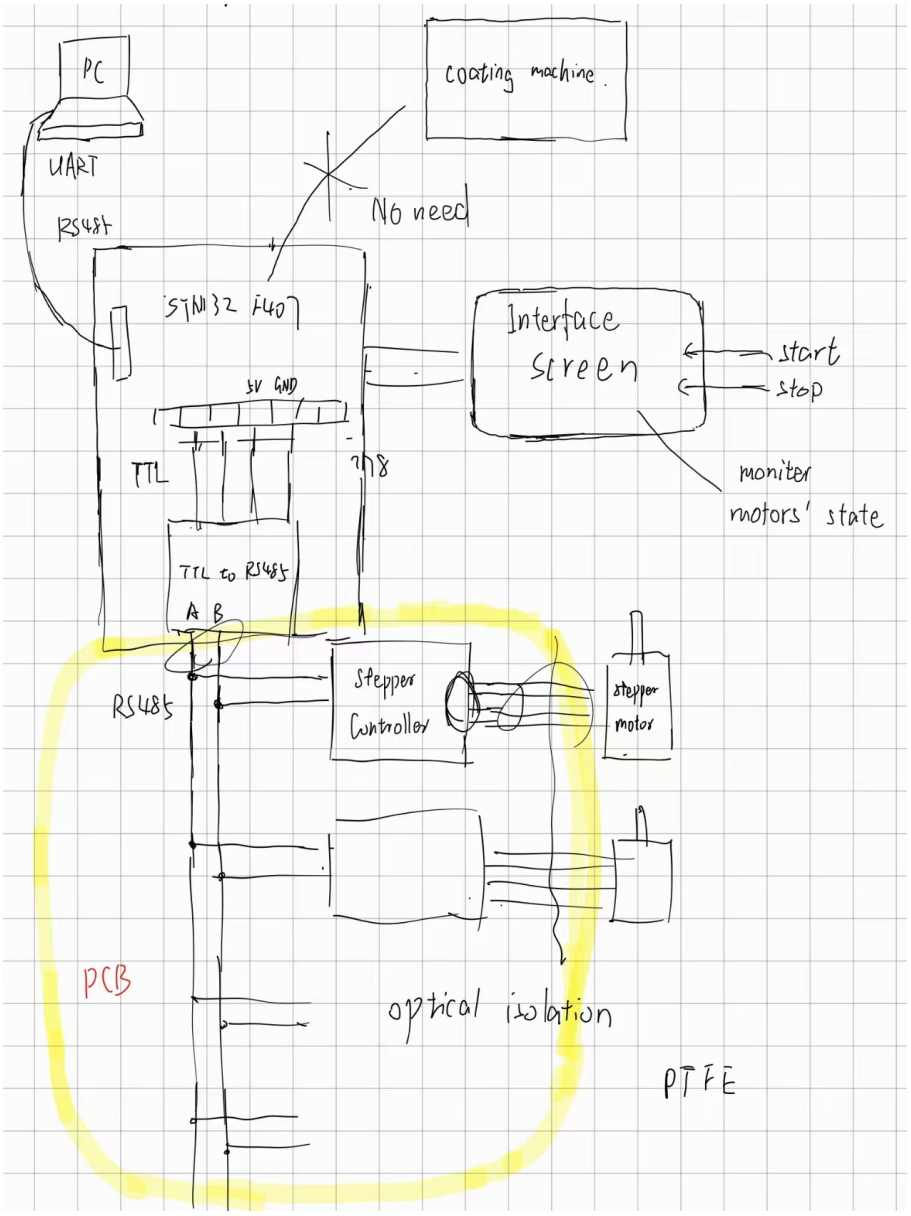


Figure 6: Draft of Control System

The control system uses STM32F407IG as the upper computer to be the central of whole control. This MCU has ARM32 Cortex-M4 CPU up to 168Mhz, able to deal with the connection with PC, touch screen, buttom and four stepper controllers. The control signals for motors uses RS485 protocol to achieve industrial-grade long range, high interference immunity, expandability.
The slave computers uses ”ZDT Emm42 stepper controller” that can deal with RS485 protocol internally. The controller can use both velocity control and position control, provided the convinience for the need of different joint. The joint 1, 2, and 3 use postion control to move the substrate to proper position. The joint 4 uses velocity control to provide constant rotation
In the MCU, runs the main program for the system (Appendix A). The class ”Motor” was built for simple storage of motor parameters and quick call-up of actions.

A Finite State Machine is used in the main sequence of program to drive the system moving to the correct position in the correct time.

- MS1_IDLE: Idle state waiting for start
- MS1_STEP1, first step of movement that moves the substrate from idle to the sputtering position
- MS1_WAIT1, a duration of time waiting for step1 to finish
- MS1_STEP2, second step of movement that adjusts the angle constantly so that substrate is coated uniformly
- MS1_WAIT2, a duration of time waiting for step2 to finish
- MS1_STEP3, move the substrate back to home position
- MS1_DONE: movment finished

2.3.2 PCB Design

As shown in Figure 7, this PCB schematic integrates a power supply and multiple RS485 communication buses. And Figure 8 shows the PCB layout diagram. Within each block, our general circuit forms are:

- Power-input block: a 1×2 pluggable terminal for 24 V in, feeding an internal 40 A copper pour, decoupled by 10 µF/50 V MLCCs placed within 5 mm of each connector.
- RS-485 breakout block: an onboard MAX3485 half-duplex transceiver drawn into a differential-pair fanout—four equal-length branches—with A/B signal control via the STM32's GPIO.
- Connector blocks: four 1×8 right-angle headers (we populate only four pins: V+, GND, A, B) arranged so that the two outer pins carry termination resistors when installed.

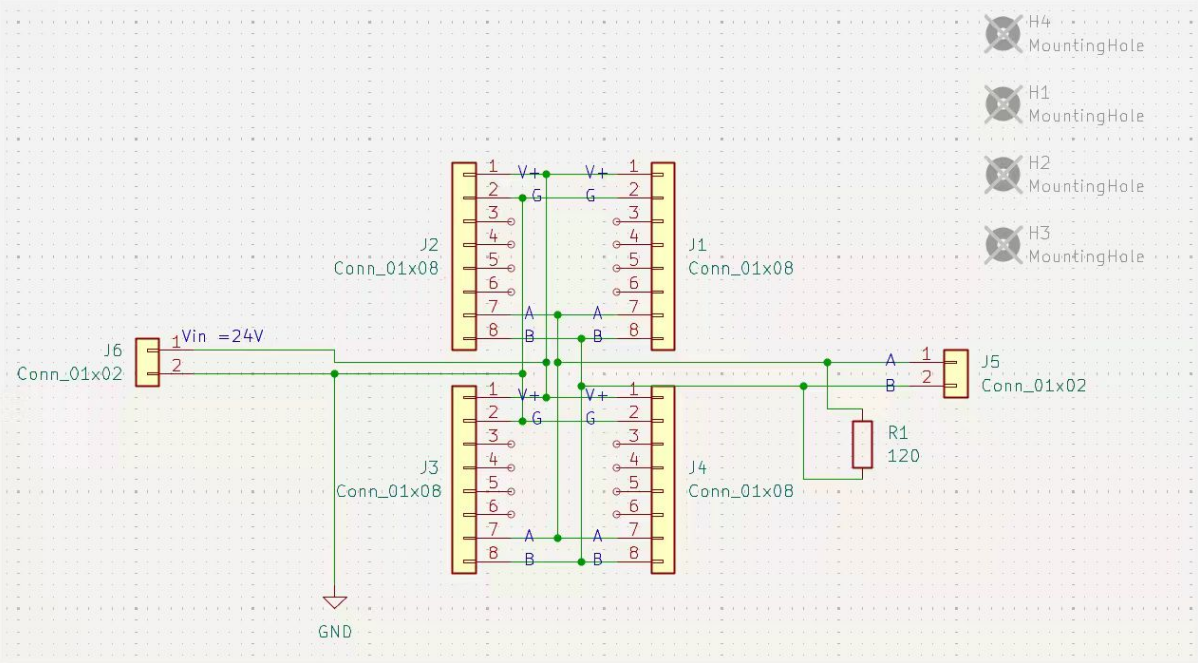


Figure 7: PCB Schematics

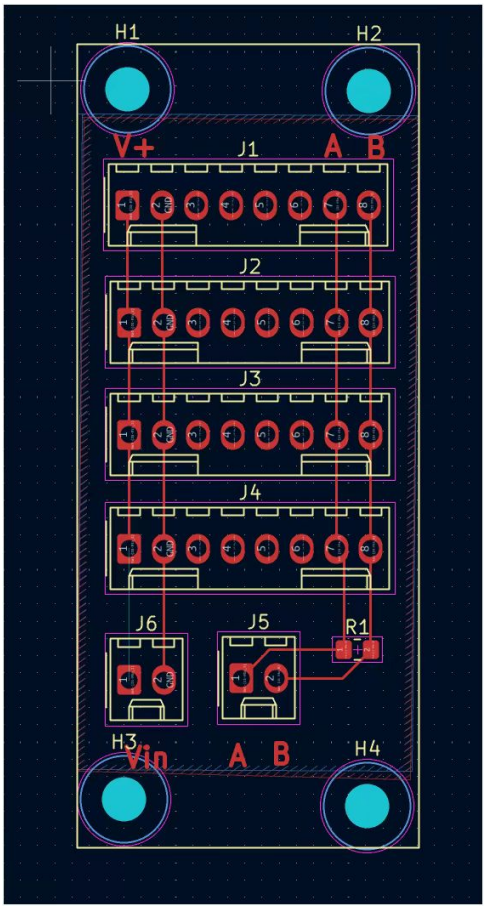


Figure 8: PCB layout

2.3.3 User Interface

(1) Buttons:

- Key 0 is assigned to execute the first complete motor-drive routine, which includes a sequential sweep of Arm 1 through Arm 4 along their full travel ranges, followed by a coordinated return to the home positions. This preset motion profile is optimized for coating cylindrical or regularly shaped specimens.
- Key 1 triggers the second distinct motor-drive routine, in which each arm follows an alternating oscillation pattern at differing amplitudes and phases—ideal for non-uniform or irregularly shaped objects requiring more complex nano-coating trajectories.
- Key 2 serves as an immediate "panic" or interruption command: upon pressing it, all ongoing motor movements are halted safely and the system enters an idle state until a new motion command is issued.

(2) TFT Screen Display: Figure 9 shows the TFT sreen display in a physical setup. Here are the detailed code implementations.

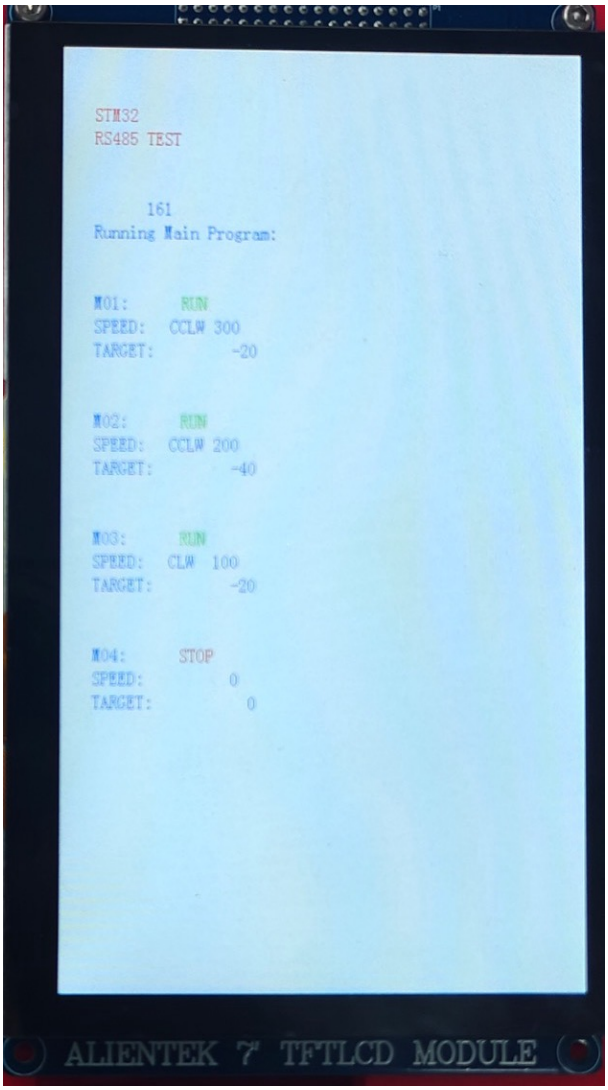


Figure 9: TFT Screen Display in the Physical Setup

- Message Area
  - Function: displayMessage(const char* msg)
  - Region: defined by

```
#define MSG_X    30
#define MSG_Y    150
#define MSG_W    500
#define MSG_H    100
```

  - Implementation: Each new message overwrites the previous one within a fixed 20-character box.

```
char buf[32];
snprintf(buf, sizeof(buf), "%-20s", msg);
lcd_show_string(MSG_X, MSG_Y, MSG_W, MSG_H, 16, buf, BLUE);
```

- Global State Display
  - Buffer:

```
char stateBuf[16];
snprintf(stateBuf, sizeof(stateBuf), "MS1:%-6s", MoveState1Names
    [(int)move1_state]);
lcd_show_string(10, 10, 200, 16, 16, stateBuf, BLUE);
```

  - What shows: “MS1:STEP1 ” (always exactly 6 chars for the state) at the top-left.
- Per-Motor Status Panels
  - Class Method:

```
void Motor::displayStatus(int baseX, int baseY) const {}
```

  - Panel Layout:
    First line: motor label and “RUN”/“STOP”

```
snprintf(buf,   , "M%02X:", addr);
lcd_show_string(baseX, baseY, 60,16,16, buf, BLUE);
// then status:
snprintf(buf,   , "%-4s", (velocity!=0)?"RUN":"STOP");
lcd_show_string(baseX+offset, baseY, 60,16,16, buf, color);
```

    Second line: “SPEED:” label + direction+value

```
lcd_show_string(..., "SPEED:", BLUE);
// dirStr = "CLW"/"CCLW" based on sign, velocity
snprintf(displayBuf, , "%-4s%3d␣", dirStr, abs(velocity));
lcd_show_string(..., displayBuf, BLUE);
```

Third line: "TARGET:" label + target angle

```
lcd_show_string(..., "TARGET:", BLUE);
snprintf(buf,   , "%5d", (int)tgt_degree);
lcd_show_string(..., buf, BLUE);
```

- Periodic Poll and Update
  - Trigger:

```
static uint8_t t = 0;
if (++t >= 20) {
    t = 0;
    // send read commands:
    for (addr=1 4 ) {
      Emm_V5_Read_Sys_Params(addr, S_VEL);
      Emm_V5_Read_Sys_Params(addr, S_CPOS);
      Emm_V5_Read_Sys_Params(addr, S_FLAG);
    }
}
```

  - Decode Refresh:

    In "Translatereceiveddata()", we parse incoming RS-485 frames and update each motor's readvelocity, readdegree, and reachpos, then immediately call each motor's displayStatus() to repaint that panel.

2.3.4 First Edition of Mechanical Design

A specialized aluminum alloy (Al) joint assembly has been designed to interface the robotic arm with the coating machine's central axis. Leveraging Al's high strength-to-weight ratio and vacuum compatibility, the joint features precision-machined surfaces for coaxial alignment with both the machine's axis and the arm's structural components. Engineered to withstand multi-axis dynamic stresses while maintaining dimensional stability under vacuum, the joint undergoes surface treatments to enhance corrosion resistance and minimize particle generation, ensuring compliance with nanocoating purity requirements. This component enables seamless torque transmission and precise specimen positioning relative to the sputtering source, while its design prioritizes CNC manufacturability for cost-effective production.

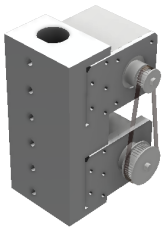


Figure 10: The Design of The First Joint

Building upon the primary structure design of the first joint, its assembly integrates a reducer and a stepper motor. This configuration not only facilitates CNC machining but also delivers sufficient torque to actuate the second joint, its connecting link, and subsequent aluminum components.

2.3.5 The Design of the First Link

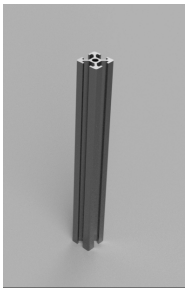


Figure 11: The Design Of The First Link

To connect the third joint while meeting strength and cost requirements, the linkage must support motor, joint, and specimen weights without excessive expense. CNC machining is unsuitable due to high costs for the required length, so a 2020 aluminum extrusion tube is optimal. This modular AL tube offers sufficient stiffness, a 20x20mm profile, and pre-engineered T-slots for easy assembly, balancing structural integrity and affordability. The Design of the Second Joint

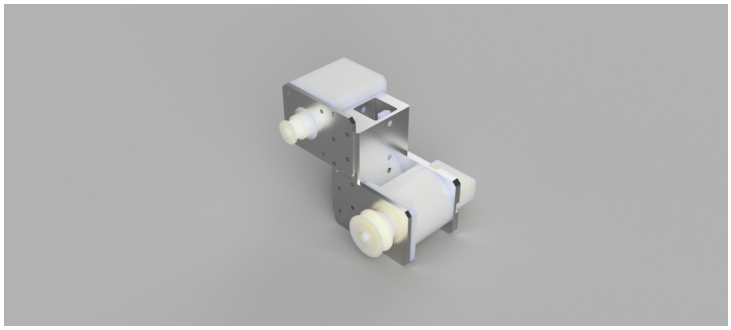


Figure 12: The Design Of The Second Joint

The second joint's primary structure adheres to a design philosophy similar to the first, featuring a stress-optimized aluminum alloy frame. However, the perpendicular arrangement of the first and second links necessitates a mirrored configuration for the stepper motor and reducer, which are mounted on opposing lateral faces of the main body. This layout optimizes torque transmission, balances inertial loads, and facilitates modular assembly, ensuring high-precision operation across all motion profiles.

2.3.6 The Design of the Third Joint

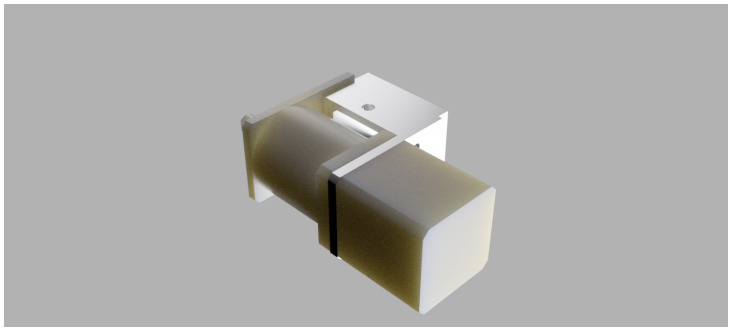
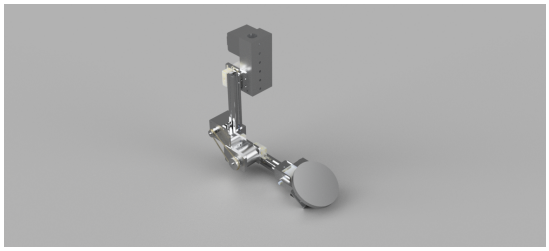


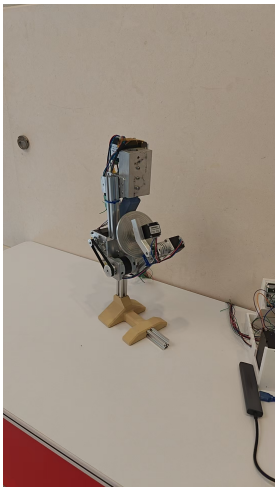
Figure 13: The Design Of The Third Joint

This component is designed to secure 2020 aluminum profiles and enable rotation of the coating platform mounted at the distal end. Given the low mass of the end-effector platform, the connection does not require a torque-enhancing reducer, allowing for a simplified structural design.

2.3.7 The two Versions of the Robotic Arm



(a) The first generation of the robotic arm



(b) The first Generation of the Robotic Arm in real world

Figure 14: Physical Design of the robotic arm 1

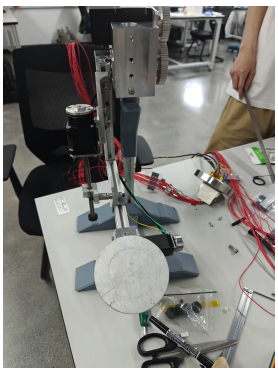
After fabricating the three-joint robotic arm prototype, manual range-of-motion testing identified critical structural and mechanical flaws:

1.Excessive Flexure in Links: Significant horizontal deflection and instability in Link 2 and Link 3 were observed, caused by the inadequate cross-sectional modulus of 2020 aluminum extrusions. This flexure undermines positioning accuracy and induces vibrations during dynamic operation.

2.Torque Deficiency in Joint 2: A payload test at the end-effector caused immediate slippage in the PLA gearing system at Joint 2 (shoulder). Analysis shows the direct-drive configuration lacks sufficient torque multiplication to overcome inertial forces.

3.Environmental Degradation of Belt Drives: Synchronous belts exhibited premature wear and material degradation in magnetron muttering's vacuum and high-temperature environment, requiring a shift to chemically inert materials or sealed transmission systems.

(a) The second Generation of the Robotic Arm



(b) The second generation of the robotic arm in the real world

Figure 15: Physical Design of the robotic arm 2

In response to the identified issues, a second-generation robotic arm prototype has been developed. Key modifications include replacing the synchronous belt drive system with a gear-driven mechanism to enhance torque transmission reliability, and integrating a motor equipped with an electromagnetic braking system to prevent unintended movement during power outages. While structural components from the first iteration—such as aluminum extrusion profiles and joint mounting interfaces—were retained for design continuity, critical drivetrain elements were upgraded to address mechanical deficiencies, like the joint 2, the original stepper motor has been replaced with a screw motor, which not only delivers more stable force but also eliminates slipping caused by the robotic arm's weight. The new design additionally enables manual adjustment of torque at the second connection. This phased redesign approach balances cost efficiency with performance improvements, ensuring compatibility with existing sub assemblies while resolving primary failure modes observed in initial testing.

3 Verification

3.1 Control Module

3.1.1 Microcontroller Unit (MCU)

Requirement	Verification
Work as the upper computer for the system. Can send RS-485 signals to stepper controllers and drive a TFT touch screen	Run a set of test programs for computation and control provided by the manufacturer, check the functionality. A. Run the ”Light and speaker test”, the light should turn on and off sequentially. B. Run the ”Touchscreen test”, the screen should work as a drawing board that can be drawn by fingers.

3.1.2 Stepper Motor Controller

Requirement	Verification
Receive RS-485 signals from the MCU and control motors correctly.	Install in the system and run test programs, the motor should be controlled to the velocity or the position we assigned. A. Send ”01 F6 01 00 64 0A 00 6B”, the motor should run at 100RPM. B. Send ”01 FD 01 05 DC 00 00 00 7D 00 00 00 6B”, the motor should run 10 rounds.

3.2 Actuator Module

3.2.1 Stepper Motor

Requirement	Verification
Three 42 stepper motors and one screw motor are needed. They need to actuate the robotic arm stably (The torque should be able to bear the weight of the robotic arm)	A. Running the simulations in MATLAB and Fusion 360 to make sure that the torques of the stepper motors should be larger than the torques needed to bear the weight. B. Install the system and mark the start position, run a simple up and down program 100 times, the end position should be within 1mm tolerance.

3.2.2 Reduction Gear

Requirement	Verification
The reduction gear need to bear the weight of the robotic arm and exert enough torque to actuate the manipulator when combined working with the stepper motors.	A. Theoretical calculation to make sure the Torqueis enough. B. Install the reduction gears on the arm. Set the motor to run at half the rated current(redundancies). The motor can move smoothly within the set range of motion.

3.3 Mechanical Arm Structure

Requirement	Verification
1. The jack screw mechanism should be compatible with the other three step motors. During the prismatic motion of connector along the slideway of the jack screw, the second link of the robotic arm should revolute smoothly around the first joint. 2. The weight of the robotic arm should be uniformly distributed. This is to decrease the required torque to actuate the manipulator and avoid fracture or cracks. 3. The length of each link of the robotic arm should be carefully considered. It should be guaranteed that there is no interference between the trajectory of the robotic arm and the working space.	1. A CAD model will be constructed in Fusion 360, and the compatibility of the steo motors and jack screw can be verified by conducting the actuation simulation of the whole robotic arm. 2. The weight distribution of the robotic arm and the resultant torques, followed by safety factor, concentrated stress can be guaranteed by finite element analysis(FEA) 3. The robotic arm will be modelled in Matlab Simulink. The trajectory of the robotic arm will be visualized using Simulink to see if there is interference with the working space

3.4 PCB Design

Requirement	Verification
1.Deliver a stable 24 V line capable of 3 A continuous per branch with less than 0.1 V drop at the farthest connector. 2.Maintain a half-duplex RS-485 differential pair between the STM32 and each motor controller, with proper DE/RE control and 120 Ω termination at both ends.	1. Apply 24 V at 3 A into each branch in turn and measure the voltage at every motor-driver connector to confirm the drop stays below 0.1 V. 2. Hook the A/B lines from the MCU to a single motor controller, transmit a known test frame (e.g. a CRC-checked position or velocity command), and confirm correct reception and response on the controller side.

3.5 Interface Module

3.5.1 button

Requirement	Verification
1. Start Button: Pressing the button initiates the robotic arm's planned trajectory. 2. Stop Button: Pressing the button immediately halts all robotic arm motion.	1. Power on the system, press the start button and observe robotic arm motion. Robotic arm is expected to begin moving along the predefined trajectory if the start button is working properly. 2. Start the robotic arm, press the stop button during motion and observe robotic arm behavior. Robotic arm is expected to stop all motion immediately (no residual movement) if the stop button is working properly.

3.5.2 TFT touch screen

Requirement	Verification
1. Display text correctly 2. touch works	1. Run the "Text Display test", 2. Run the "Touchscreen test", the screen should work as a drawing board that can be drawn by fingers. A. The line should be generated at a position where the finger touches. B. All areas of the screen can be touched and lines can be generated.

3.6 Circuit Connection

3.6.1 CF63 Conflat Flang Interface

Requirement	Verification
The 4*4 cables needed by 4 motors are connected through the interface, signaling well.	Connect cables through it, resistor should be below $1\Omega for every cable$.

3.6.2 Teflon Insulated Cable

Requirement	Verification
Carry signals and power from CF63 to motors safely, not being damaged by the sputtering or signal interfered.	A. The technical specs will be checked to match the nano coating machine before buy it. The cable should be resistant to more than 300 degrees. B. Install it in the sputtering machine, control the motor during the sputtering process. Check the cables after one coating process, the cable should not have visual impairment.

4 Costs

Development cost can be estimated to be 50 RMB/hour. Four members in the group work from Dec/2024 to May/2025, 5 hours/week for first 3 months, 12 hours/week for last 3 months.

$$50[RMB] * (5[hrs] * 12[weeks] + 12[hrs] * 13[weeks]) = 10800$$

4.1 Prototype

Part	Price(RMB)	Quantities	Cost (RMB)
Arduino UNO R3	169	1	169
Stepper controller	90	4	360
STM32F407ZGT6 Development Board	658.52	1	658.52
”Emm42” stepper controller	60	6	360
42mm Stepper motor	38	2	76
42mm reduction gear (1:50)	120	1	120
42mm reduction gear (1:10)	100	1	100
28mm Stepper motor	45	2	90
28mm reduction gear (1:10)	145	1	145
screw motor	598	1	598
2020 aluminum profile	N/A	5	21.62
Half moon shaped cast aluminum base	7.45	1	7.45
Customized aluminum parts	N/A	N/A	2591
24V power supply	118.9	1	118.9
CF63 connector	1200	1	1200
Teflon insulated cable	5.68	1	56.8

In the process of prototype development, we tried different components to get better performance. Some materials that are already available in the laboratory, such as cables, high temperature tape, and aluminum foil, were used in the development process, so they were not included in the cost.

4.2 bulk

For the mass production, the cost will be lower.

Part	Price(RMB)	Quantities	Cost (RMB)
STM32F407ZGT6 Development Board	658.52	1	658.52
”Emm42” stepper controller	60	4	240
42mm Stepper motor	38	1	38
42mm reduction gear (1:50)	120	1	120
28mm Stepper motor	45	2	90
28mm reduction gear (1:10)	145	1	145
42 screw motor	598	1	598
2020 aluminum profile	N/A	2	5
Half moon shaped cast aluminum base	7.45	1	7.45
Customized aluminum parts	N/A	N/A	2591
24V power supply	118.9	1	118.9
CF63 connector	1200	1	1200
Teflon insulated cable	5.68	10m	56.8
Regular cable	NA	18*5m	≈ 50

The Development Board can be replaced with a board designed only for this use, so that cost can be even lower.

5 Conclusion

The iterative design culminated in a robust system featuring four motors, reducers, a microcontroller, and custom mechanics, including a screw-driven joint for enhanced torque and stability. Verification of mechanical, control (STM32-based with RS-485), and interface modules confirmed operational functionality, preparing the stage for definitive coating and tribo-testing experiments to demonstrate its superiority.

Key accomplishments include the full mechanical and control system design (including a custom PCB), assembly, and initial functional verification. The project effectively iterated from initial concepts to a refined V2 design, overcoming early prototype limitations like torque deficiency and material degradation. The system now offers precise multi-axis manipulation within a vacuum, controlled via a TFT screen and button interface.

While core functionality is proven, comprehensive long-term vacuum endurance and extensive coating trials on diverse geometries are the next steps to fully quantify performance gains and optimize motion profiles. Contingency plans for material outgassing or payload limitations involve sourcing UHV-specific components or refining mechanical elements. Cost reduction for potential scaling is also feasible through more specialized PCB design.

Ethical Considerations: Adherence to the IEEE Code of Ethics was central, particularly ensuring public welfare by aiming to improve nanocoated product quality (e.g., biomedical implants) and designing for safe lab operation. Claims regarding system capabilities are based on available data and iterative testing, reflecting honesty and realism in design and reporting.

Broader Impacts: This four-axis vacuum stage significantly impacts nano-manufacturing by enabling efficient, high-quality coating of complex 3D objects. Economically, this can reduce costs and improve product performance in biomedical, aerospace, and advanced materials sectors. Environmentally, precise coating can minimize material waste. Societally, it advances nanotechnology applications, contributing to technological progress and improved material solutions. In summary, this design project delivered a functional and innovative four-axis vacuum stage, demonstrating a practical solution for advanced 3D nanocoating and setting the stage for impactful experimental validation.

References

[1] R. A. Caruso, "Nanocasting and Nanocoating," in *Colloid Chemistry I*, A. De Meijere, H. Kessler, S. V. Ley, J. Thiem, F. Vögtle, K. N. Houk, J.-M. Lehn, S. L. Schreiber, B. M. Trost, H. Yamamoto, and M. Antonietti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, vol. 226, pp. 91–118.

[2] W. Bao, Z. Deng, S. Zhang, Z. Ji, and H. Zhang, "Next-Generation Composite Coating System: Nanocoating," *Frontiers in Materials*, vol. 6, p. 72, Apr. 2019.

[3] D. H. Abdeen, M. El Hachach, M. Koc, and M. A. Atieh, "A Review on the Corrosion Behaviour of Nanocoatings on Metallic Substrates," *Materials*, vol. 12, no. 2, p. 210, Jan. 2019.

[4] G. Choi, A. H. Choi, L. A. Evans, S. Akyol, and B. Ben-Nissan, "A review: Recent advances in sol-gel-derived hydroxyapatite nanocoatings for clinical applications," *Journal of the American Ceramic Society*, vol. 103, no. 10, pp. 5442–5453, Sep. 2020.

[5] I. Shishkovsky and P. Lebedev, "Chemical and physical vapor deposition methods for nanocoatings," in *Nanocoatings and Ultra-Thin Films*. Elsevier, 2011, pp. 57–77.

[6] V. Falikman, "Nanocoatings in modern construction," *Nanotechnologies in Construction A Scientific Internet-Journal*, vol. 13, no. 1, pp. 5–11, Feb. 2021.

[7] O. V. Penkov, V. E. Pukha, S. L. Starikova, M. Khadem, V. V. Starikov, M. V. Maleev, and D.-E. Kim, "Highly wear-resistant and biocompatible carbon nanocomposite coatings for dental implants," *Biomaterials*, vol. 102, pp. 130–136, Sep. 2016.

[8] K. Kruthika, B. Kiran Kumar, and S. Lakshminarayanan, "Design and development of a robotic arm," in *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*. Bangalore: IEEE, Oct. 2016, pp. 1–4.

Appendix A - Main Program

```
#include "./c_bindings.h"
#include <math.h> //       #include <cmath>


//
enum MoveState1 {
    MS1_IDLE = 0,
    MS1_STEP1,  //
    MS1_WAIT1,  //       30s
    MS1_STEP2,  //
    MS1_WAIT2,  //       60s
    MS1_STEP3,  //
    MS1_DONE    //
};

static MoveState1 move1_state = MS1_IDLE;
static uint32_t move1_lastTick = 0;

const char* MoveState1Names[] = {
    "IDLE",
    "STEP1",
    "WAIT1",
    "STEP2",
    "WAIT2",
    "STEP3",
    "DONE"
};

//
#define MSG_X    30
#define MSG_Y    150
#define MSG_W    500
#define MSG_H    100

//
void displayMessage(const char* msg) {
    //                                  20
    char buf[32];
    snprintf(buf, sizeof(buf), "%-20s", msg);
    // "%-20s"                    20

    lcd_show_string(MSG_X, MSG_Y, MSG_W, MSG_H, 16, buf, BLUE);
}


```

```
46
47
48
49
50  class Motor {
51  private:
52      uint8_t addr;           // Motor address
53      uint8_t set_dir;        // Set forward direction (0 or 1)
54      uint8_t dir;            // Current direction bit sent to the motor
55      uint8_t redu_ratio;     // Reduction ratio of the motor gearbox
56      uint8_t acc;            // Acceleration parameter for the motor
57      uint16_t vel;           // Target velocity (raw value to be sent to
            motor)
58
59  public:
60      int velocity;           // User-defined speed (RPM)
61      double tgt_degree;      // Target angle (degrees)
62      int32_t read_velocity;  // Real-time read velocity (RPM)
63      int32_t read_position_raw;// Real-time read raw position count (
            signed)
64      double read_degree;     // Real-time read angle (degrees)
65      bool reach_pos;
66
67      // Initializes the motor address, direction, reduction ratio, and
            acceleration
68      void init(uint8_t address, uint8_t direction = 0, uint8_t
            reduction_ratio = 1, uint8_t acc_val = 10) {
69          addr = address;
70          set_dir = direction;
71          redu_ratio = reduction_ratio;
72          acc = acc_val;
73          velocity = 0;
74          tgt_degree = 0;
75          read_velocity = 0;
76          read_position_raw = 0;
77          read_degree = 0;
78      }
79
80      // Calculates the absolute position count value after setting the
            target angle
81      uint32_t read_tgt_pos() const {
82          return static_cast<uint32_t>(tgt_degree * (3200.0 * redu_ratio) /
                360.0);
83      }
84
85      // Sets the target position (in degrees) and optionally the velocity
            , then sends the position control command
```

```
86      uint32_t tgt_position(double degree, int velocity_val = 1) { //
            [/3200 = round] - Comment likely referring to a rounding aspect
            in the underlying implementation
87          tgt_degree = degree;
88          velocity = velocity_val;
89          if (degree < 0) {
90              dir = set_dir ? 0 : 1; // Reverse direction if the degree is
                    negative
91              degree = -degree; // Store the absolute value of the degree
92          } else {
93              dir = set_dir;
94          }
95          uint32_t position = degree * (3200 * redu_ratio) / 360; // [/3200
                = round] - Comment likely referring to a rounding aspect in
            the underlying implementation
96          if (velocity_val!=0) {
97              vel = static_cast<uint16_t>(velocity_val); // [RPM] - Set
                    velocity if a velocity value is provided
98          }
99          Emm_V5_Pos_Control(addr, dir, vel, acc, position, 0, 0); // [
                degree] - Send position control command to the motor
100         return position;
101     }
102
103     // Sets the velocity and direction based on the input velocity value
            , without sending a command
104     void set_velocity(int velocity_val) {
105         velocity = velocity_val;
106         if (velocity < 0) {
107             vel = static_cast<uint16_t>(-velocity); // Store the absolute
                    value of the negative velocity
108             dir = set_dir ? 0 : 1;            // Reverse direction if the
                     velocity is negative
109         } else {
110             vel = static_cast<uint16_t>(velocity); // Store the positive
                    velocity
111             dir = set_dir;                     // Maintain the set
                    forward direction
112         }
113     }
114
115     // Sends the constant speed control command to the motor
116     void constant_rorate() {
117         Emm_V5_Vel_Control(addr, dir, vel, acc, 0);
118     }
119
120     void constant_rorate(int velocity_val) {
121         set_velocity(velocity_val);
```

```
122         Emm_V5_Vel_Control(addr, dir, vel, acc, 0);
123     }
124 
125     // Returns the reduction ratio of the motor
126     uint8_t get_reduction_ratio() const { return redu_ratio; }
127 
128     // Displays the current status (speed, target angle, working status)
        on the TFT screen
129     void displayStatus(int baseX, int baseY) const {
130         char buf[32];
131         const int offsetX = 10; // Horizontal spacing
132         int currentY = baseY;
133         int currentX = baseX;
134 
135         // --- First line: Motor label and status (STOP/RUN) ---
136         // Output status with a fixed width of 4 characters
137         snprintf(buf, sizeof(buf), "M%02X:", addr);
138         lcd_show_string(currentX, currentY, 60, 16, 16, buf, BLUE);
139         currentX += 60 + offsetX * 2;
140 
141         const char* status = (velocity != 0) ? "RUN" : "STOP";
142         uint16_t color = (velocity != 0) ? GREEN : RED;
143         // Fixed width of 4 characters, padded with spaces on the right
144         snprintf(buf, sizeof(buf), "%-4s", status);
145         lcd_show_string(currentX, currentY, 60, 16, 16, buf, color);
146 
147 
148          // --- Second line: SPEED and DIRECTION or STOP "0" ---
149         currentY += 20;
150         currentX = baseX;
151         lcd_show_string(currentX, currentY, 60, 16, 16, "SPEED:", BLUE);
152         currentX += 60 + offsetX;
153 
154         // --- Second line: SPEED and DIRECTION or STOP "0" ---
155         currentY += 20;
156         currentX = baseX;
157         lcd_show_string(currentX, currentY, 60, 16, 16, "SPEED:", BLUE);
158         currentX += 60 + offsetX;
159 
160         char displayBuf[10];
161         if (velocity > 0) {
162             const char* dirStr = (set_dir == 0) ? "CLW" : "CCLW";
163             //                  8    4                  + 3           +
                1
164             snprintf(displayBuf, sizeof(displayBuf), "%-4s%3d␣", dirStr,
                velocity);
165 
166         } else if (velocity < 0) {
```

```
167             const char* dirStr = (set_dir == 0) ? "CCLW" : "CLW";
168             snprintf(displayBuf, sizeof(displayBuf), "%-4s%3d␣", dirStr, -
                    velocity);
169
170         } else {
171             //       0              8
172             snprintf(displayBuf, sizeof(displayBuf), "␣␣␣␣0␣␣␣");
173         }
174         //
175         lcd_show_string(currentX, currentY, 70, 16, 16, displayBuf, BLUE);
176
177         // --- Third line: TARGET DEGREE ---
178         currentY += 20;
179         currentX = baseX;
180         lcd_show_string(currentX, currentY, 100, 16, 16, "TARGET:", BLUE);
181         currentX += 100 + offsetX;
182         // Target angle fixed width 5 digits (including sign)
183         int tgt_int = static_cast<int>(tgt_degree);
184         snprintf(buf, sizeof(buf), "%5d", tgt_int);
185         lcd_show_string(currentX, currentY, 80, 16, 16, buf, BLUE);
186     }
187 };
188
189 // Global definitions for four motor instances
190 Motor motor1, motor2, motor3, motor4;
191
192
193 //
194 static uint32_t motorStartTick[4] = {0, 0, 0, 0}; //
                              HAL_GetTick()
195 static uint32_t motorDuration[4] = {0, 0, 0, 0}; //
                     ms
196 static bool    motorMoving[4]    = {false, false, false, false}; //
197
198
199 //                    RPM
200 static uint32_t estimateTimeMs(double degrees, int rpm) {
201     // ms = |deg|/RPM * (60*1000ms) / 360deg
202     return (uint32_t)(fabs(degrees) / rpm * 60000.0 / 360.0);
203 }
204
205 void pollMotorStops() {
206     uint32_t now = HAL_GetTick();
207     for (int i = 0; i < 4; ++i) {
208         if (motorMoving[i] && now - motorStartTick[i] >= motorDuration[i])
                {
209             //
```

```
            switch (i) {
                case 0: motor1.constant_rorate(0); break;
                case 1: motor2.constant_rorate(0); break;
                case 2: motor3.constant_rorate(0); break;
                case 3: motor4.constant_rorate(0); break;
            }
            motorMoving[i] = false; //
        }
    }
}



// Parses the received RS485 data, updates the corresponding motor, and refreshes the display
void Translate_received_data(uint8_t* rs485buf) {
    uint8_t len;
    rs485_receive_data(rs485buf, &len);
    if (len == 0) return;
    if (len > 8) len = 8;

    uint8_t motor_addr  = rs485buf[0];
    uint8_t function_code = rs485buf[1];
    Motor* pm;
    switch (motor_addr) {
        case 1: pm = &motor1; break;
        case 2: pm = &motor2; break;
        case 3: pm = &motor3; break;
        case 4: pm = &motor4; break;
        default: return;
    }

    switch (function_code) {
        case 0x35: // Velocity feedback
            if (len >= 6) {
                uint8_t sign = rs485buf[2];
                uint16_t speed_raw = (rs485buf[3] << 8) | rs485buf[4];
                pm->read_velocity = (sign == 0x01) ? -static_cast<int32_t>(speed_raw) : speed_raw;
            }
            break;
        case 0x36: // Position feedback
            if (len >= 8) {
                uint8_t sign = rs485buf[2];
                uint32_t pos = (rs485buf[3] << 24) | (rs485buf[4] << 16) |
                               (rs485buf[5] << 8) | rs485buf[6];
                pm->read_position_raw = (sign == 0x01) ? -static_cast<int32_t>(pos) : pos;
```

```
255                 uint8_t rr = pm->get_reduction_ratio();
256                 pm->read_degree = pm->read_position_raw * 360.0 / (3200.0
                        * rr);
257             }
258             break;
259         case 0x3A: // Status flag
260             if (len >= 4) {
261                 uint8_t status = rs485buf[2];
262                 pm->reach_pos = (status & 0x02) ? true : false; // Check
                        if the target position is reached
263             }
264             break;
265         default:
266             break;
267     }
268 
269     // The four motors are arranged vertically, each refreshed
            individually
270     int baseX = 30;
271     for (uint8_t i = 0; i < 4; ++i) {
272         int y = 210 + i * 100;
273         switch (i + 1) {
274             case 1: motor1.displayStatus(baseX, y); break;
275             case 2: motor2.displayStatus(baseX, y); break;
276             case 3: motor3.displayStatus(baseX, y); break;
277             case 4: motor4.displayStatus(baseX, y); break;
278         }
279     }
280 }
281 
282 //
283 void process_move_set_1(void) {
284     uint32_t now = HAL_GetTick();
285     switch (move1_state) {
286         case MS1_IDLE:
287             //
288             break;
289 
290         case MS1_STEP1:
291             //
292             motor1.tgt_position(15, 30); // [degree]
293             motorStartTick[0] = HAL_GetTick();
294             motorDuration[0] = estimateTimeMs(15, 30);
295             motorMoving[0]   = true;
296             delay_ms(10);
297             motor2.tgt_position(30, 20);
298             motorStartTick[1] = HAL_GetTick();
299             motorDuration[1] = estimateTimeMs(30, 20);
```

```
            motorMoving[1]   = true;
            delay_ms(10);
            motor3.tgt_position(40, 10); // [degree]
            motorStartTick[2] = HAL_GetTick();
            motorDuration[2] = estimateTimeMs(40, 10);
            motorMoving[2]   = true;
            delay_ms(10);
            motor4.constant_rorate(50);
            delay_ms(10);
            move1_lastTick = now;
            move1_state = MS1_WAIT1;
            break;

        case MS1_WAIT1:
            //       30   30000    ms
            if (now - move1_lastTick >= 30000) {
                move1_state = MS1_STEP2;
            }
            break;

        case MS1_STEP2:
            //        :
            motor1.tgt_position(5, 3);
            motorStartTick[0] = HAL_GetTick();
            motorDuration[0] = estimateTimeMs(5,3);
            motorMoving[0]   = true;
            delay_ms(10);
            motor2.tgt_position(10, 2);
            motorStartTick[1] = HAL_GetTick();
            motorDuration[1] = estimateTimeMs(10, 2);
            motorMoving[1]   = true;
            delay_ms(10);
            motor3.tgt_position(-20, 5);
            motorStartTick[2] = HAL_GetTick();
            motorDuration[2] = estimateTimeMs(-20, 5);
            motorMoving[2]   = true;
            delay_ms(10);
            move1_lastTick = now;
            move1_state = MS1_WAIT2;
            break;

        case MS1_WAIT2:
            //          60   60000    ms
            if (now - move1_lastTick >= 60000) {
                move1_state = MS1_STEP3;
            }
            break;

```

```
348         case MS1_STEP3:
349             //         :       home
350             motor1.tgt_position(-20, 30);
351             motorStartTick[0] = HAL_GetTick();
352             motorDuration[0] = estimateTimeMs(-20, 30);
353             motorMoving[0]   = true;
354             delay_ms(10);
355             motor2.tgt_position(-40, 20);
356             motorStartTick[1] = HAL_GetTick();
357             motorDuration[1] = estimateTimeMs(-40, 20);
358             motorMoving[1]   = true;
359             delay_ms(10);
360             motor3.tgt_position(-20, 10);
361             motorStartTick[2] = HAL_GetTick();
362             motorDuration[2] = estimateTimeMs(-20, 10);
363             motorMoving[2]   = true;
364             delay_ms(10);
365             motor4.constant_rorate(0);
366             delay_ms(10);
367             move1_state = MS1_DONE;
368             break;
369
370         case MS1_DONE:
371             //                                IDLE       DONE     Key0
372             move1_state = MS1_IDLE;
373             break;
374     }
375 }
376
377 //                move_set_2
378 void move_set_2() {
379     motor1.tgt_position(5, 4);
380     motorStartTick[0] = HAL_GetTick();
381     motorDuration[0] = estimateTimeMs(-5, 4);
382     motorMoving[0]   = true;
383     // delay_ms(10);
384     // motor2.tgt_position(5, 10);
385     // delay_ms(10);
386     // motor3.tgt_position(-20, 10);
387     // delay_ms(10);
388     // motor4.tgt_position(20,10);
389 }
390
391 //
392 int main(void) {
393     uint8_t key, t = 0, cnt = 0;
394     uint8_t rs485buf[8];
```

```
395     char stateBuf[16];    // <<
396
397     HAL_Init();
398     sys_stm32_clock_init(336, 8, 2, 7);
399     delay_init(168);
400     usart_init(115200);
401     led_init();
402     lcd_init();
403     key_init();
404     rs485_init(115200);
405
406     lcd_show_string(30, 50, 200, 16, 16, "Senior Design:", RED);
407     //
408     lcd_show_string(
409     30,       // x
410     70,       // y
411     320,      //
412     16,       //                          font_size
413     16,       //
414     "Four-Axis Vacuum Stage",
415     RED);
416
417     //
418     lcd_show_string(
419     30,       // x
420     90,       // y          font_size          +
421     320,      //
422     16,
423     16,
424     "for Advanced Nano-Manufacturing",
425     RED
426     );
427
428     //
429     motor1.init(1, 1, 100, 128);
430     motor2.init(2, 1, 30, 128);
431     motor3.init(3, 0, 10, 64);
432     motor4.init(4, 0, 1,  1);
433
434     while (1) {
435         //
436         key = key_scan(0);
437
438         if (key == KEY0_PRES && move1_state == MS1_IDLE) {
439             displayMessage("Running Main Program");
440             move1_state = MS1_STEP1;    //
441         }
442         if (key == KEY1_PRES && move1_state == MS1_IDLE) {
```

```
443             displayMessage("Running Program 2");
444             move_set_2();
445         }
446         if (key == KEY2_PRES) {
447             displayMessage("STOP");
448             //
449             motor1.tgt_position(0,0);
450             delay_ms(10);
451             motor2.tgt_position(0,0);
452             delay_ms(10);
453             motor3.tgt_position(0,0);
454             delay_ms(10);
455             motor4.tgt_position(0,0);
456 
457             move1_state = MS1_IDLE; //
458         }
459 
460 
461         process_move_set_1();       //
462         pollMotorStops();          //
463 
464         //                          6            "WAIT2"    "STEP1"
465         //                             6
466         char stateBuf[16];
467         snprintf(stateBuf, sizeof(stateBuf), "MS1:%-6s", MoveState1Names[(
                int)move1_state]);
468         //                          "IDLE" (4     )
469         //
470 
471         lcd_show_string(10, 10, 200, 16, 16, stateBuf, BLUE);
472 
473         //            RS485
474         if (++t >= 20) {
475             t = 0;
476             LED0_TOGGLE();
477             cnt++;
478             lcd_show_xnum(78, 130, cnt, 3, 16, 0x80, BLUE);
479             for (uint8_t addr = 1; addr <= 4; ++addr) {
480                 Emm_V5_Read_Sys_Params(addr, S_VEL);
481                 Emm_V5_Read_Sys_Params(addr, S_CPOS);
482                 Emm_V5_Read_Sys_Params(addr, S_FLAG);
483             }
484         }
485 
486         Translate_received_data(rs485buf);
487 
488         //
```

```
        HAL_Delay(1);
    }

    return 0;
}
```

Appendix B - Engineering Drawing

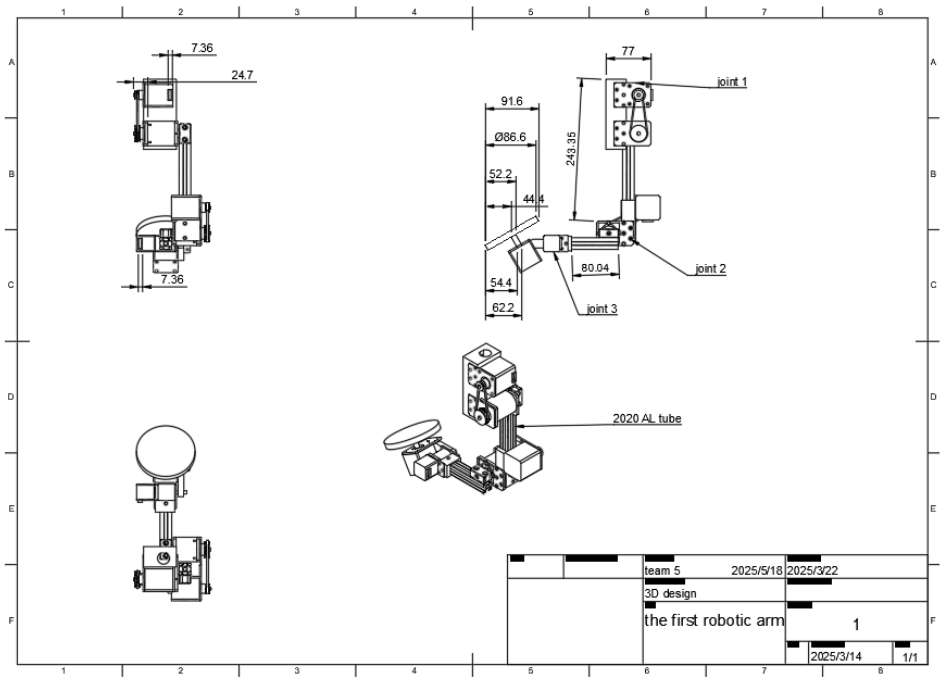


Figure 16: The Engineering Drawing of version one

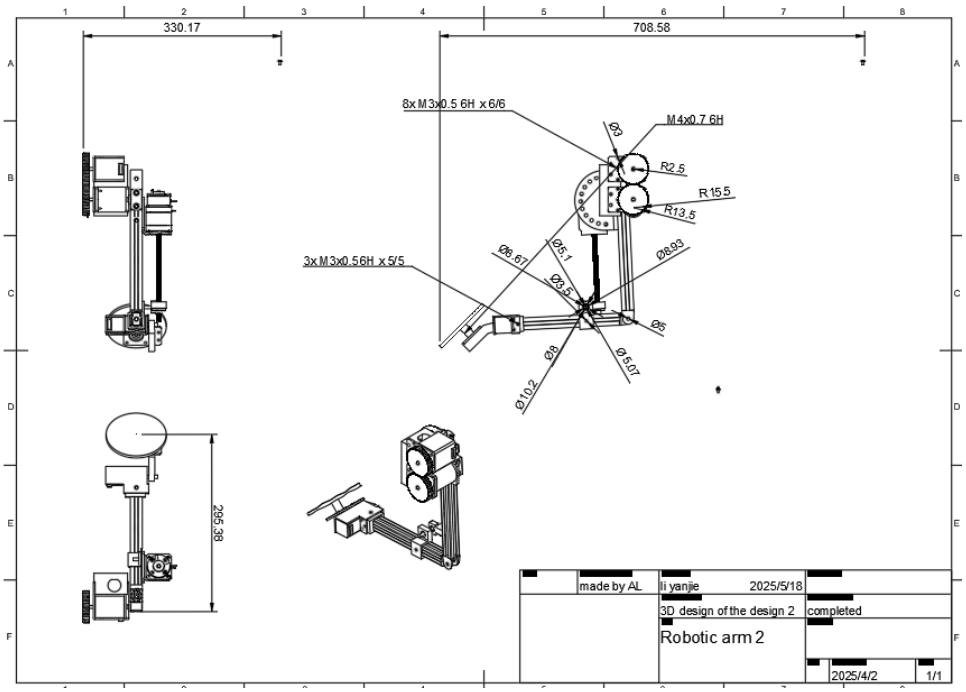


Figure 17: The Engineering Drawing of version one