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.

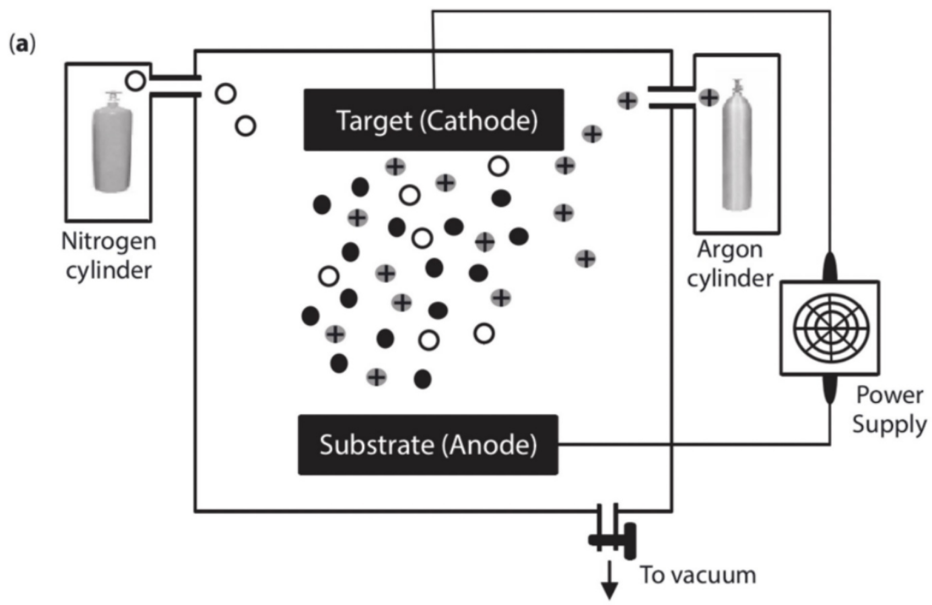


Figure 1: A schematic of magnetron sputtering process

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:** $_{d}iagram_{j}oint1.png$`[Block Diagram: MCU -> Driver -> Stepper -> Reducer -> Joint1 Output]`
$Function{:}Controllersignalsdrivethemotorviaadriver; thereducerincreasestorque forarmbaserotation.$

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 ($T_{req} \propto mgl \cdot \mathrm{SF}$, 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.

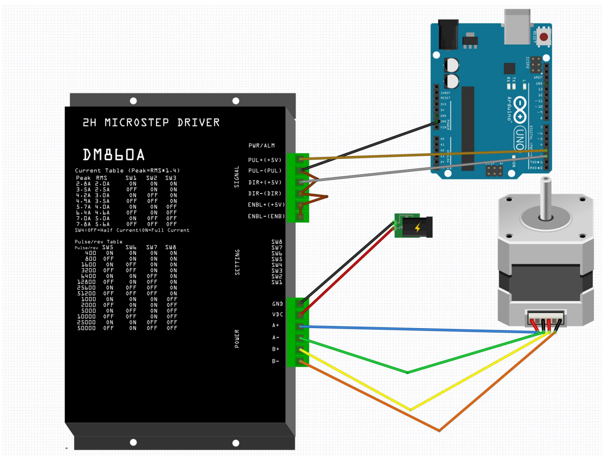


Figure 2: One Unit of Stepper Connection

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).

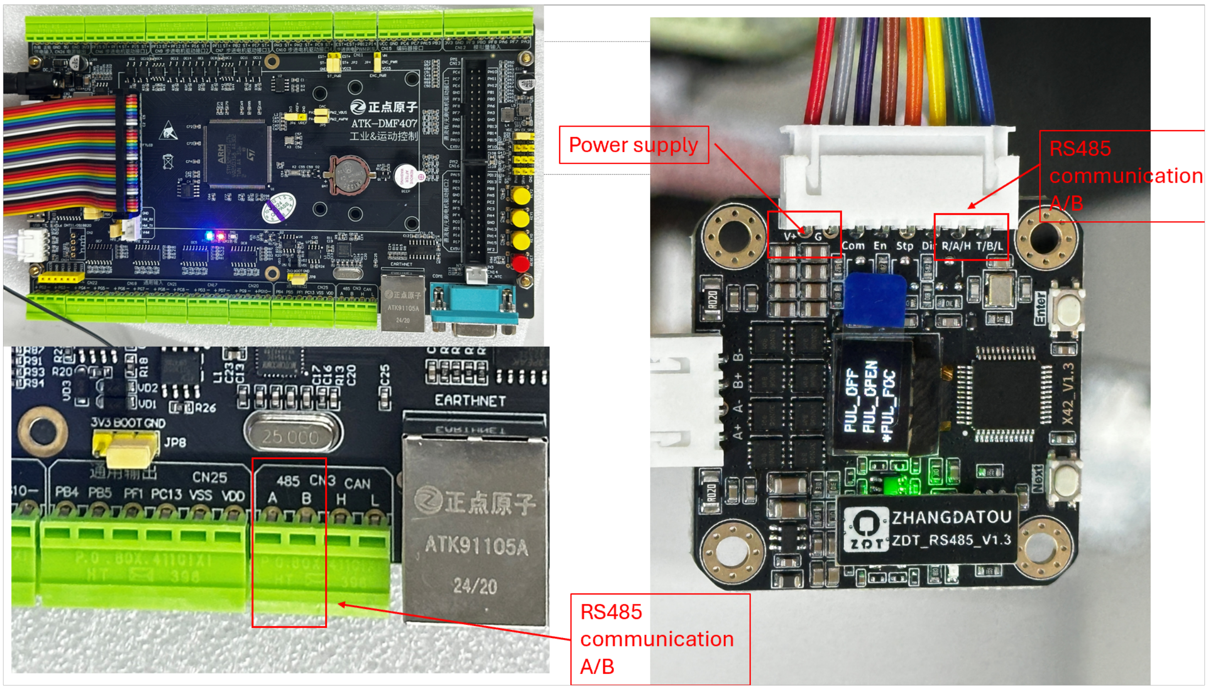


Figure 3: Close-up View of MCU and Motor Driver

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.

2.2.1 Dynamic analysis of electrical motors

Table 2: Component Weights for Torque Calculations

Component	**Spec/Type**	**Weight (approx.)**
42*48 stepper motor	0.6Nm	350g
28*30 stepper motor	0.07Nm	100g
Screw Motor	0.06 mm/step	250g
42*51 reduction gear	50:1	350g
28*33 reduction gear	10:1	200g
Shaft - Joint 1 connection	Aluminum	570g
Joint 1 - 2 connection	Aluminum	190g
Joint 2 - 3 connection	Aluminum	45g
End-effector Plate	r=50mm, Aluminum	109g

2.2.2 Calculation

Joint-4: plate rotation:

To calculate the moment of inertia, the formula is shown below:

$$I_{plate} \approx \frac{1}{2} \times M \times r^2 \tag{1}$$

After calculation, the moment of inertia is $1.36 \times 10^{-4} [kg \cdot m^2]$

To calculate the angular acceleration, we have:

$$\alpha = \frac{\tau}{I} \approx \frac{0.07Nm}{1.36 \times 10^{-4}} = 513.76[rad \cdot s^{-2}] \tag{2}$$

Joint-3: claw:

$$\tau_{claw-maxpower} = \tau_{stepper} \times GR \tag{3}$$

Where $\tau_{claw\_maxpower}$ is the maximum power that the claw can reach, $\tau_{stepper}$ is the applied torque of the step motor and GR is the reduced ratio of the claw reducer.

$$\tau_{claw\_maxpower} = 0.07Nm \times 10 = 0.7[N \cdot m]$$

$$\tau_{claw} \approx (109g + 100g) \times 35mm \approx 0.0073[N \cdot m] << \tau_{claw\_maxpower}$$

Joint-2: forward arm:

Using the same formula demonstrated in joint-3, we can calculate and compare the exerted and required torques

$$\tau_{farm\_maxpower} = 0.6N \cdot m \times 2 \times 10 = 12[N \cdot m]$$

$$\tau_{farm} \approx (109g + 100g) \times 130mm + (45g + 100g + 200g) \times 100mm \approx 0.062[N \cdot m]$$

After comparison, we can find that $\tau_{farm\_maxpower} >> \tau_{farm}$

Joint-1: back arm:

With the same method, we have:

$$\tau_{barm\_maxpower} = 0.6N \cdot m \times 2 \times 50 = 60[N \cdot m]$$

$$\begin{gathered}\tau_{barm} \approx \\ (109g + 100g) \times 200mm + (45g + 100g + 200g) \times 170mm + (190g + 350g + 250g) \times 70mm \\ \approx 0.16[Nm] << \tau_{barm\_maxpower}\end{gathered}$$

All joint torque load at a safe range.

2.3 Design details and accomplishments

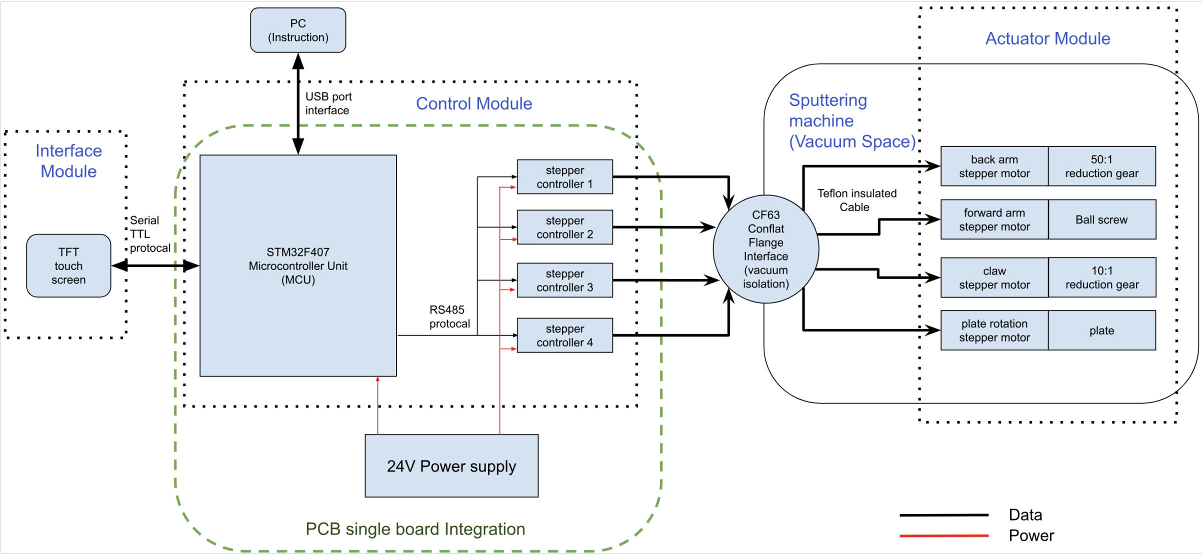


Figure 5: Block diagram of the system

2.3.1 Control System

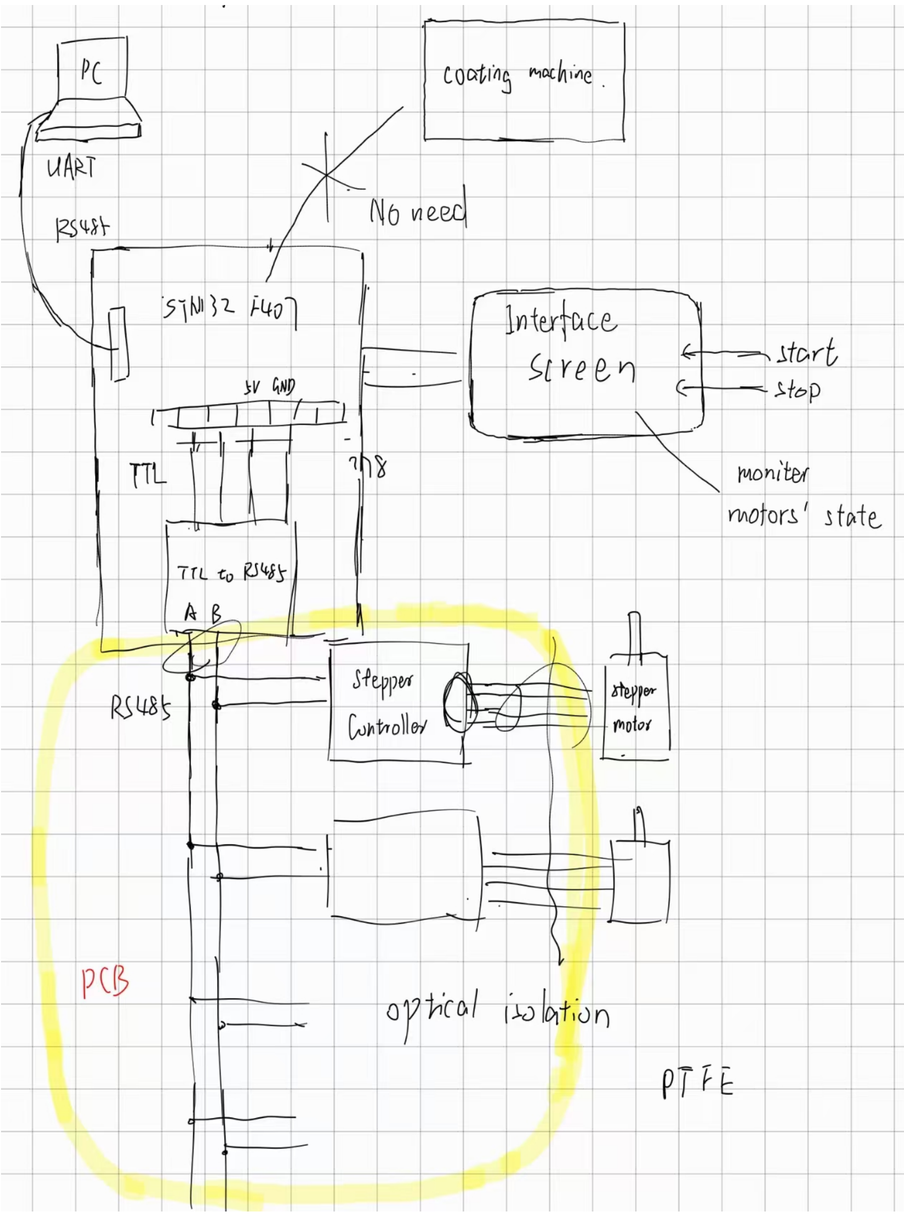


Figure 6: Draft of Control System

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.

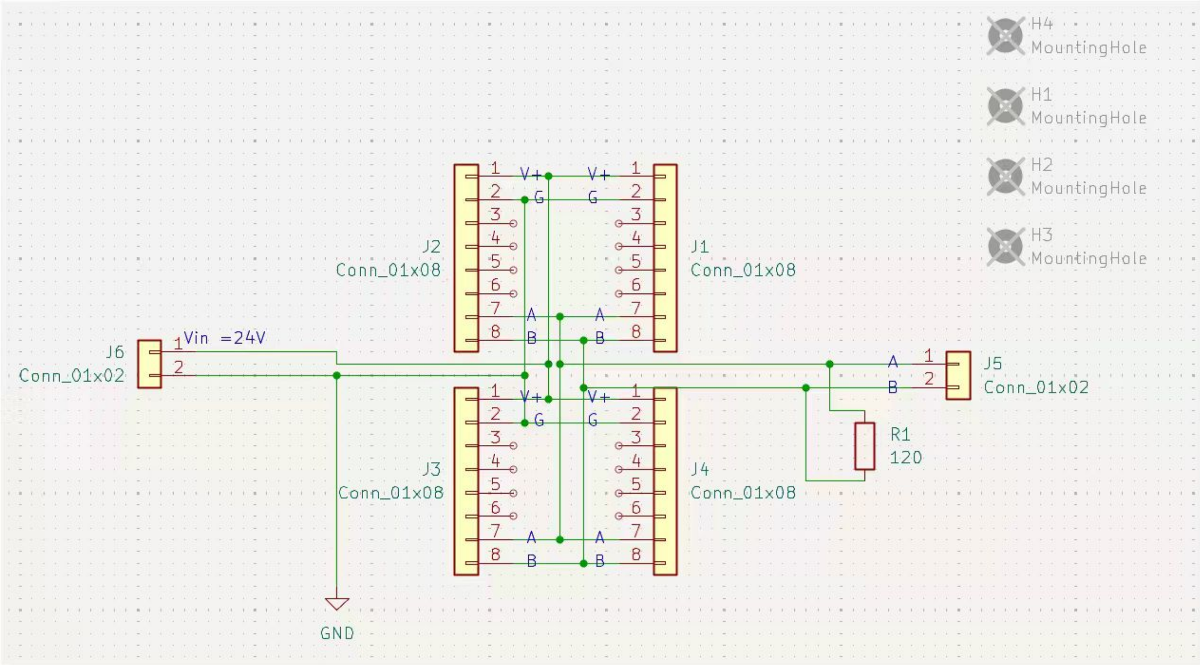


Figure 7: PCB Schematics

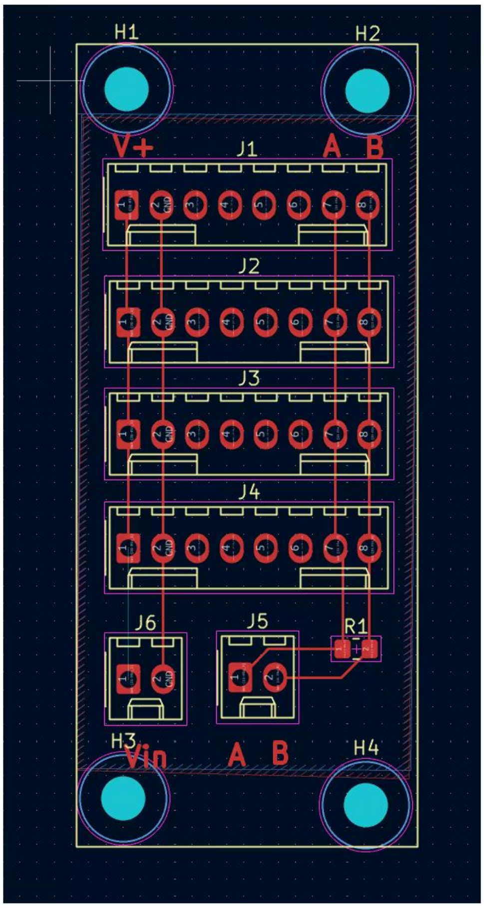


Figure 8: PCB layout

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.

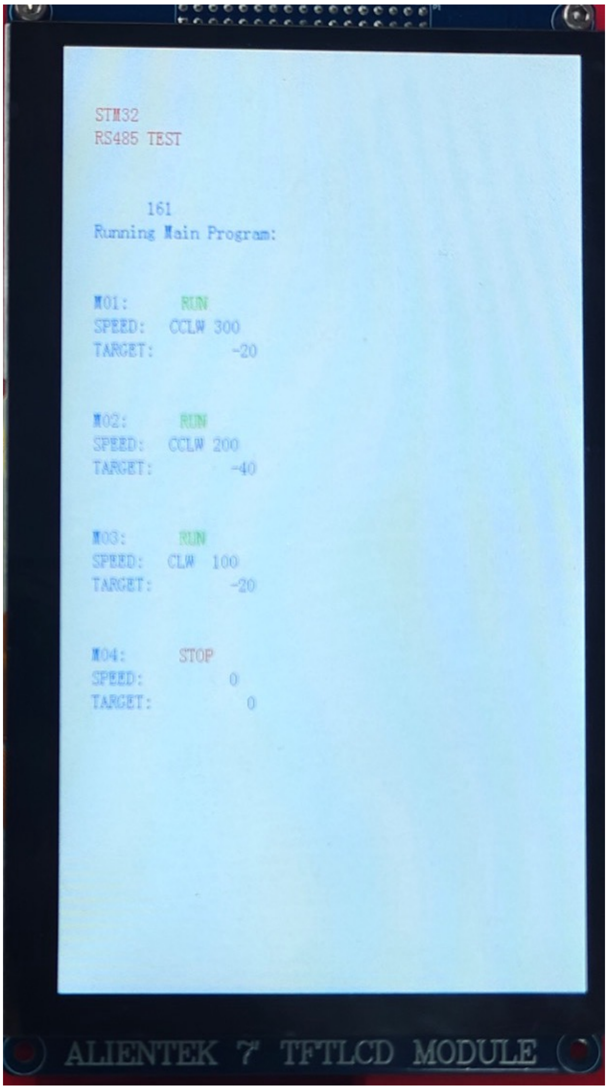


Figure 9: TFT Screen Display in the Physical Setup

- 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.

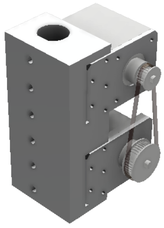


Figure 10: The Design of The First Joint

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

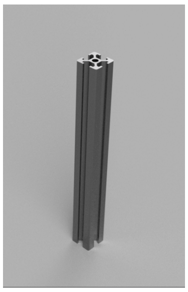


Figure 11: The Design Of The First Link

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

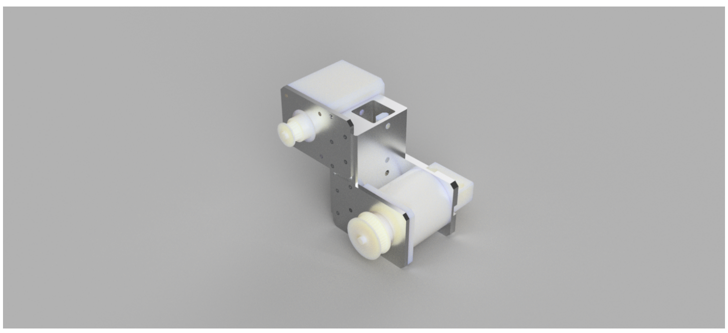


Figure 12: The Design Of The Second Joint

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

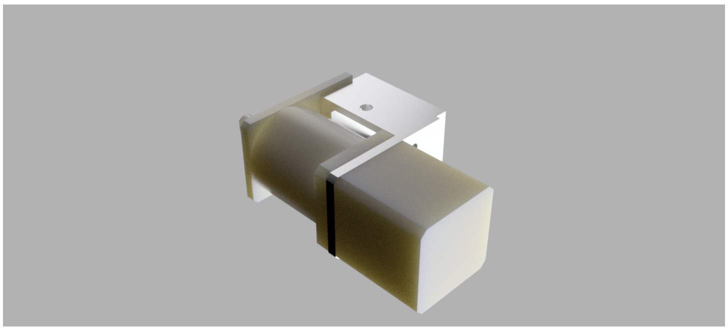
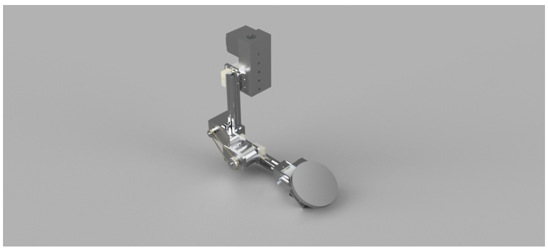


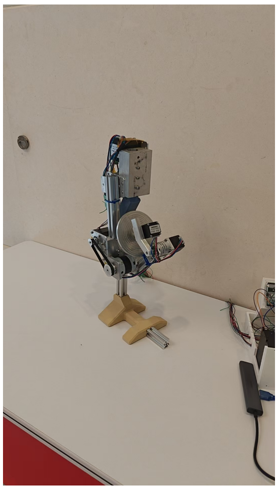
Figure 13: The Design Of The Third Joint

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

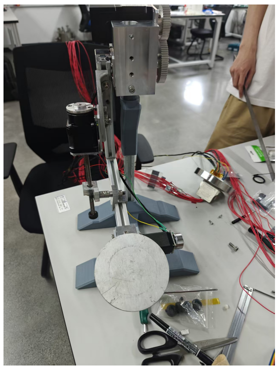
After fabricating the three-joint robotic arm prototype, manual range-of-motion testing identified critical structural and mechanical flaws:

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

This section details the verification procedures undertaken to ensure that individual components and the integrated system meet the design requirements. Each requirement was tested, and the results are summarized below.

3.1 Control Module

3.1.1 Microcontroller Unit (MCU)

Requirement	Verification Method	Result/Status
The MCU must function as the upper computer, capable of sending RS-485 signals to stepper controllers and driving the TFT touchscreen.	A set of manufacturer-provided test programs for computation and control were executed on the STM32 board. A. The ”Light and speaker test” was run. B. The ”Touchscreen test” was run.	**Passed.** A. **Passed.** Lights turned on and off sequentially, and the speaker functioned as expected. B. **Passed.** The screen correctly registered touch inputs, allowing lines to be drawn by finger across the entire active area.

3.1.2 Stepper Motor Controller

Requirement	Verification Method	Result/Status
The ”ZDT Emm42” stepper controllers must receive RS-485 signals from the MCU and accurately control motor velocity and position.	Controllers were integrated into the system. Specific RS-485 command sequences were sent from the MCU to test motor responses. A. Command ”01 F6 01 00 64 0A 00 6B” (target: 100 RPM) was sent. B. Command ”01 FD 01 05 DC 00 00 00 7D 00 00 00 6B” (target: 10 rotations) was sent.	**Passed.** A. **Passed.** The connected motor ran at the commanded 100 RPM. B. **Passed.** The connected motor completed 10 full rotations as commanded.

3.2 Actuator Module

3.2.1 Stepper Motor

Requirement	Verification Method	Result/Status
The three 42-stepper motors and one screw motor must actuate the robotic arm stably, with torque sufficient to bear the arm's weight and payload. Positional repeatability should be within 1mm.	A. Torque adequacy was assessed via simulations in MATLAB and Fusion 360, considering component weights (ref. Table 2).	A. **Confirmed.** Simulations indicated that the selected motors provide torque exceeding the calculated static and dynamic requirements.
	B. The fully assembled system's repeatability was tested. A start position was marked, and a program executing a simple up-and-down motion was run for 100 cycles. The final end position was compared to the initial mark.	B. **Passed.** After 100 cycles, the deviation of the end position was observed to be [e.g., "0.X mm"], which is within the 1mm tolerance. This confirmed stable actuation and satisfactory repeatability under the tested conditions.

3.2.2 Reduction Gears

Requirement	Verification Method	Result/Status
Reduction gears must reliably transmit torque from the stepper motors to actuate the manipulator joints and bear the weight of subsequent arm segments and payload.	A. Theoretical torque calculations were performed, incorporating gear ratios and motor torques.	A. **Confirmed.** Calculations showed that the output torque from the gear-motor assemblies is sufficient for the anticipated loads.
	B. Reduction gears were installed on the arm. Motors were operated at half their rated current (to test for sufficient torque margin). Smoothness of motion within the full range was observed.	B. **Passed.** The motors successfully drove the joints smoothly throughout their intended range of motion even at reduced current, indicating adequate torque transmission and a margin of safety.

3.3 Mechanical Arm Structure

Requirement	Verification Method	Result/Status
1. The jack screw mechanism (Joint 2) must be compatible with the overall arm kinematics, allowing smooth revolution of Link 2.	1. A CAD model of the complete V2 robotic arm was constructed in Fusion 360. Actuation simulations were performed, focusing on the jack screw and its interaction with connected links.	1. **Passed.** CAD simulations demonstrated smooth revolute motion of Link 2, driven by the prismatic motion of the jack screw's connector, without mechanical interference.
2. The weight of the robotic arm must be distributed to minimize excessive torque requirements and avoid stress concentrations leading to fracture or cracks.	2. Finite Element Analysis (FEA) was conducted on the CAD model to analyze weight distribution, resultant torques, safety factors, and stress concentrations under static and dynamic load conditions.	2. **Confirmed.** FEA results showed that stress concentrations were within material limits (e.g., maximum stress of [X] MPa for Aluminum 6061-T6) and safety factors (e.g., minimum SF of [Y]) were adequate, validating the structural integrity.
3. Link lengths must be appropriate to ensure the arm's trajectory does not interfere with the defined working space or internal components of the coating machine.	3. The robotic arm kinematics were modeled in Matlab Simulink. Trajectories for typical coating operations were simulated and visualized.	3. **Passed.** Simulink trajectory visualizations confirmed that the arm operates within the designated workspace without collisions under the simulated motion paths.

3.4 PCB Design

Requirement	Verification Method	Result/Status
1. Deliver a stable 24V power line to each of the four motor drivers, capable of 3A continuous per branch, with less than 0.1V drop at the farthest connector.	1. The custom PCB was powered. Each power branch was loaded with 3A in turn, and the voltage was measured at every motor-driver connector using a digital multimeter.	1. **Passed.** The maximum measured voltage drop at the farthest connector under a 3A load was [e.g., ”0.0X V”], which is below the 0.1V threshold.
2. Maintain a half-duplex RS-485 differential pair communication between the STM32 and each motor controller, with proper DE/RE control and 120 Ω termination.	2. The A/B lines from the MCU’s RS-485 transceiver on the PCB were connected to a single motor controller. Known test frames (CRC-checked position/velocity commands) were transmitted. Signal integrity was observed using an oscilloscope, and response from the controller was monitored. This was repeated for all four channels.	2. **Passed.** Correct reception and response were confirmed for each motor controller. Oscilloscope readings showed clean differential signals, and DE/RE timing was appropriate, ensuring reliable communication.

3.5 Interface Module

3.5.1 Button Functionality

Requirement	Verification Method	Result/Status
1. Start Button (KEY0): Pressing initiates the robotic arm’s planned trajectory.	1. The system was powered on. The Start Button (KEY0) was pressed. Robotic arm motion was observed.	1. **Passed.** The robotic arm began moving along its predefined trajectory as expected upon pressing KEY0.
2. Stop Button (KEY2): Pressing immediately halts all robotic arm motion.	2. The robotic arm was set in motion using a predefined trajectory. During motion, the Stop Button (KEY2) was pressed. Robotic arm behavior was observed.	2. **Passed.** All motion of the robotic arm stopped immediately upon pressing KEY2, with no observable residual movement.

3.5.2 TFT Touch Screen

Requirement	Verification Method	Result/Status
1. Display text and numerical data correctly and clearly.	1. A ”Text Display test” program was run, showing various strings, numbers, and status indicators on the screen.	1. **Passed.** All text and numerical data were displayed correctly, with appropriate font, color, and alignment as defined in the UI design.
2. Touch input must be accurately registered across the screen surface.	2. The ”Touchscreen test” (drawing board application) was run. Lines were drawn by touching various parts of the screen. A. Lines were generated at the precise position where the finger touched. B. All areas of the screen were responsive to touch, and lines could be generated across the entire active display.	2. **Passed.** A. **Confirmed.** B. **Confirmed.**

3.6 Circuit Connection Integrity

3.6.1 CF63 Conflat Flange Interface

Requirement	Verification Method	Result/Status
The 16 conductors (4 motors x 4 cables each) passing through the CF63 Conflat flange interface must maintain good electrical continuity for reliable signal transmission.	After assembly, the electrical resistance of each conductor path through the flange and connected cables was measured using a multimeter.	**Passed.** The measured resistance for every conductor was below 1Ω (typically [e.g., ”0.X Ω”]), confirming proper connections and low-resistance paths.

3.6.2 Teflon Insulated Cables (In-Vacuum)

Requirement	Verification Method	Result/Status
Teflon insulated cables must reliably carry signals and power from the CF63 flange to the motors inside the vacuum chamber, resisting damage from the sputtering environment (high temperature, plasma).	A. Technical specifications of the selected cables were checked against the known operating conditions of the nanocoating machine (e.g., max temperature, vacuum compatibility) before procurement.	A. **Confirmed.** Cable specifications (e.g., rated for $> 300°C$, low outgassing) were verified to be suitable for the intended environment.
	B. The cables were installed in the sputtering machine. The robotic arm was operated (motors controlled) during a standard chromium sputtering process. Cables were visually inspected after the coating process.	B. **Passed.** The motors operated without issues during the sputtering process. Post-experiment visual inspection of the Teflon cables revealed no signs of melting, charring, cracking, or other physical impairment.

3.7 Nanocoating Experiments (System-Level Functional Verification)

To verify the overall feasibility, functionality, and superiority of the vacuum stage for its intended application, a series of nanocoating experiments were conducted in the Advanced Nanocoating Lab using the magnetron sputtering machine integrated with the robotic manipulator.

3.7.1 Chromium Coating on Stainless Steel Substrate

Objective: To prove the basic feasibility of the robotic manipulator for nanocoating on a flat substrate.
Procedure: A stainless steel specimen was placed flatly on the end-effector plate of the robotic manipulator and secured with carbon tape (Fig. 17). The assembly was introduced into the magnetron sputtering machine (Fig. 16). Chromium was sputtered onto the substrate (Fig. 18) with a sputtering current of 300 mA for 0.5 hours at a set temperature of $300°C$. During deposition, the substrate was held static by the arm.
Results: A uniform chromium coating was successfully deposited on the central area of the stainless steel substrate (Fig. 19). The measured thickness of the coating was approximately 900 nm.
Conclusion: Passed. The experiment demonstrated that the four-axis vacuum stage is feasible for conducting nanocoating experiments and can produce coatings of high uniformity and quality on simple geometries.

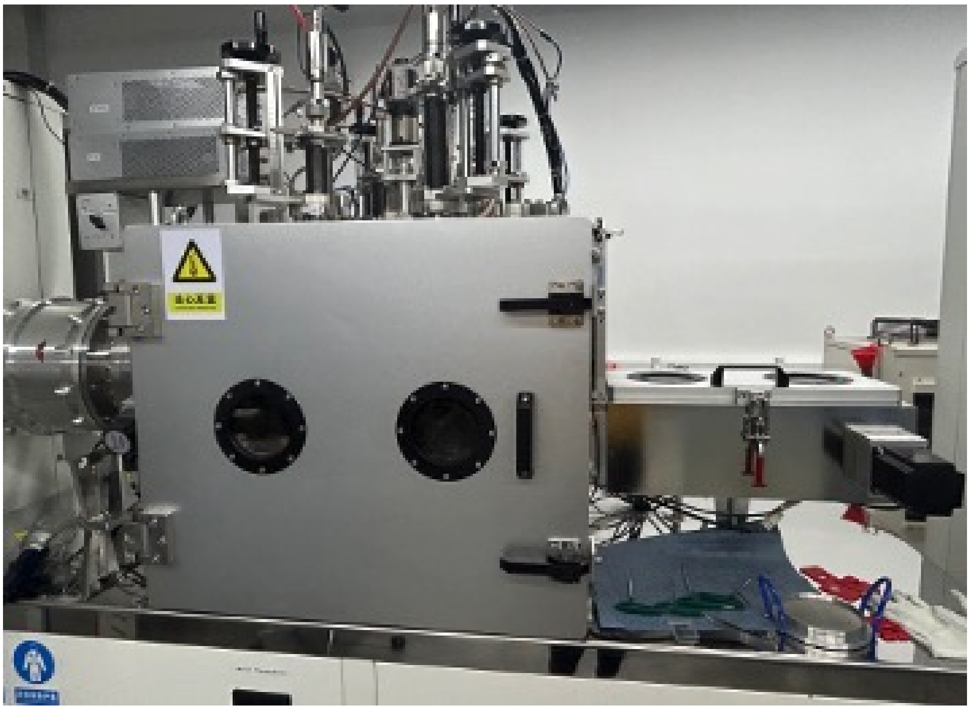


Figure 16: Magnetron sputtering machine in the Advanced Nanocoating Lab.

Figure 17: Preparation stage of the nanocoating experiment with a stainless steel substrate.

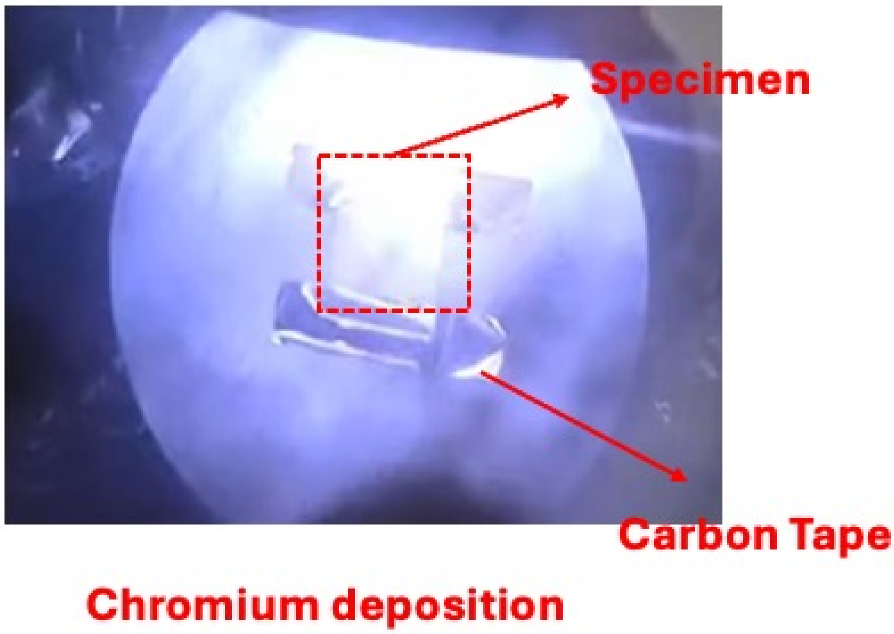


Figure 18: Chromium coating process on the stainless steel substrate using the robotic stage.

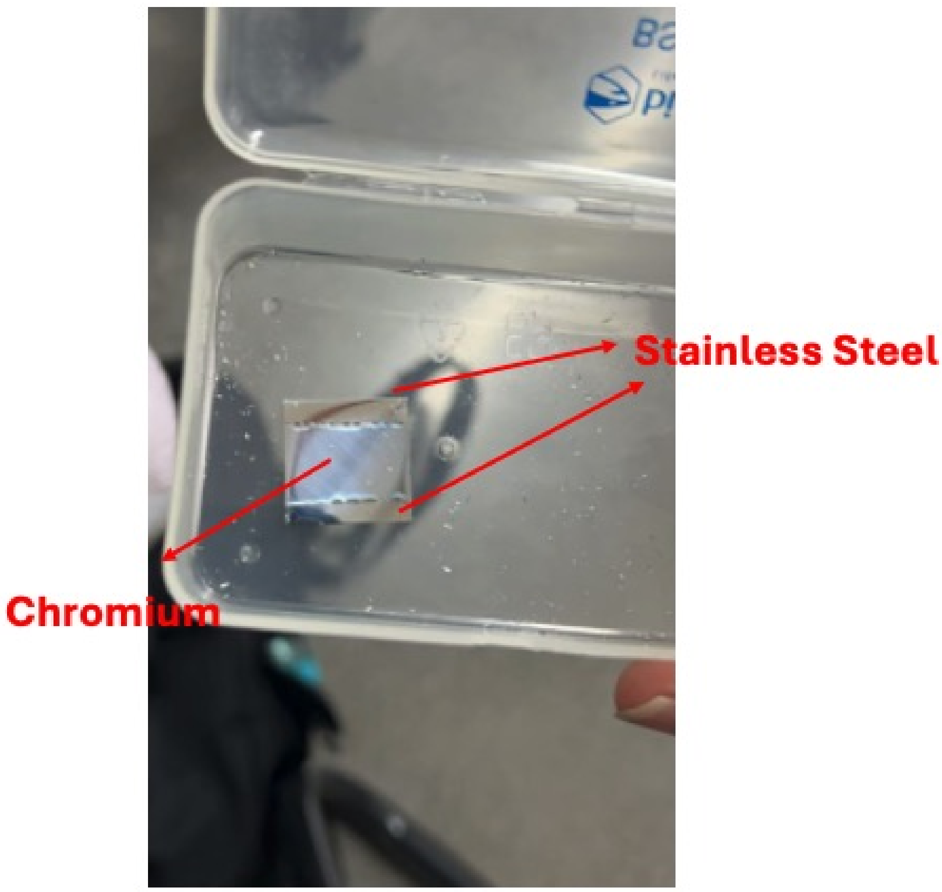


Figure 19: Result of the chromium nanocoating on the stainless steel substrate.

3.7.2 Chromium Coating on 3D Brass Cylinder

Objective: To demonstrate the robotic manipulator's capability for coating 3D objects, showcasing its superiority over static substrate holders.
Procedure: A 3D brass cylinder was mounted on the end-effector (Fig. 20). During the chromium sputtering process, the robotic arm was programmed to rotate and tilt the cylinder to expose its different surfaces to the sputtering source, aiming for a conformal coating. [Specific motion parameters, e.g., rotation speed, tilt angles, should be mentioned if available].
Results: A chromium coating was successfully applied to the surfaces of the 3D brass cylinder (Fig. 21). Visual inspection indicated coverage over the curved and flat surfaces of the cylinder. [Quantitative analysis of uniformity on the 3D object, if performed, should be mentioned here, e.g., SEM images, thickness measurements at different points].
Conclusion: Passed. This experiment successfully demonstrated the device's capability to aid in the nanocoating of 3D or irregularly shaped objects, a key objective of the project. The ability to manipulate the substrate during deposition is crucial for achieving uniform coatings on complex geometries.

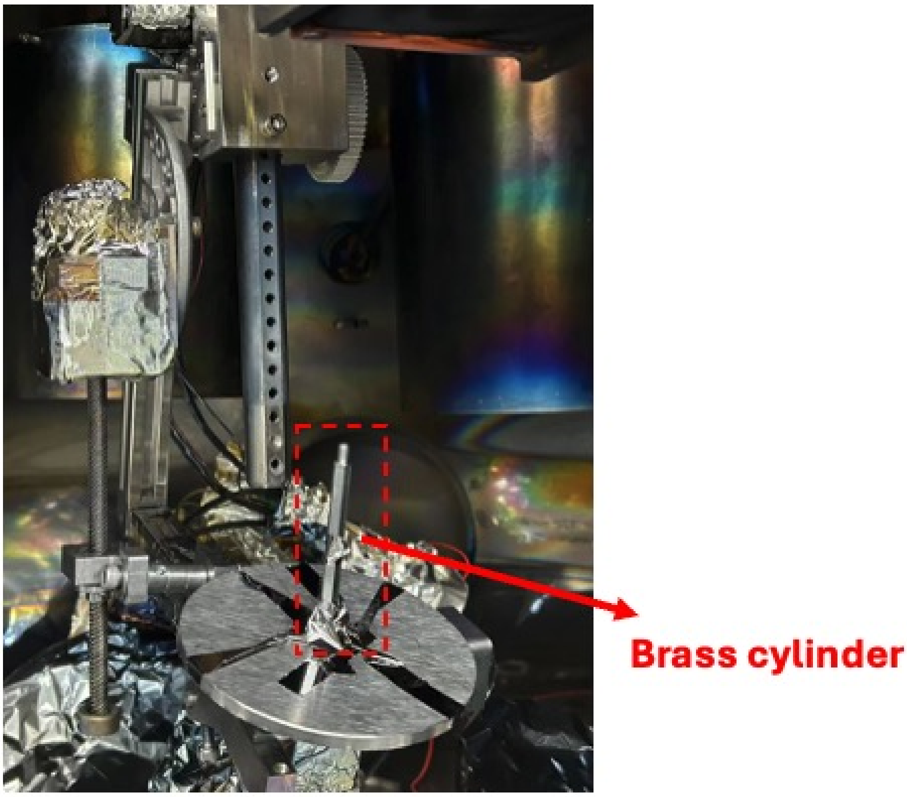


Figure 20: Preparation stage before the coating experiment for the 3D brass cylinder.

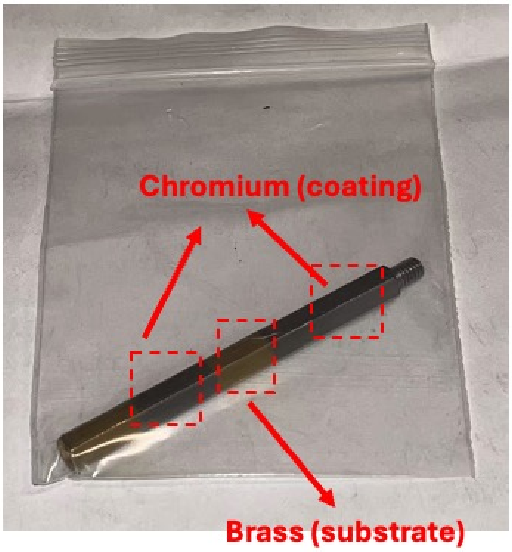


Figure 21: Result of nanocoating chromium onto the 3D brass cylinder.

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

Table 3: Prototype Development Cost Breakdown

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 Final Device

For the mass production, the cost will be lower.

Table 4: Bulk Production Cost Estimates

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 Uncertainty

The performance of any precision mechatronic system, such as the Four-Axis Vacuum Stage, is inherently limited by various sources of uncertainty. A comprehensive understanding and quantification of these uncertainties are crucial for evaluating the system's reliability, predicting its operational accuracy, and ensuring the quality of the nano-manufacturing processes it supports. This section outlines the primary sources of uncertainty identified in our design and their potential impact on the system's performance.

5.1 Sources of Uncertainty

Uncertainties in the four-axis vacuum stage can be broadly categorized into mechanical, actuation, control, and environmental/operational sources.

5.1.1 Mechanical System Uncertainties

- **Manufacturing Tolerances:** Components such as the custom aluminum joints, links, and mounting interfaces are subject to manufacturing tolerances inherent in CNC machining and fabrication processes. Typical tolerances for machined parts (e.g., ± 0.05 mm to ± 0.1 mm, *students to insert their specific tolerance data here from part specifications or measurements*) can lead to minor deviations in dimensions, perpendicularity, and parallelism. These deviations can accumulate through the kinematic chain, affecting the end-effector's precise positioning.

- **Assembly Precision:** The manual assembly of the robotic arm can introduce small misalignments between joints and links. While care is taken during assembly, slight angular or linear offsets can contribute to systematic positioning errors. *(If any specific alignment checks or resulting deviations were noted, mention them here.)*

- **Backlash:** Mechanical backlash is present in the gear reducers integrated with the stepper motors and, to a lesser extent, in the screw motor mechanism of Joint 2. Backlash can cause a small amount of lost motion when a joint reverses direction, impacting repeatability and fine positioning accuracy. *(Students should attempt to quantify this from component datasheets or empirical measurement, e.g., "Estimated backlash in geared joints is X arcmin/degrees. The screw motor datasheet specifies Y mm backlash.")*

- **Structural Deflection:** The aluminum extrusions (Link 1) and other structural components will experience some degree of elastic deformation under the weight of the arm and payload, and during dynamic movements. While FEA was considered for weight distribution (as mentioned in Verification 3.3.2), residual deflections can introduce transient and static positioning errors. The V2 design aimed to mitigate excessive flexure identified in V1. *(If FEA results provided estimated deflections, or if any deflections were observed under load, mention them.)*

5.1.2 Actuation System Uncertainties

- **Stepper Motor Resolution:** Stepper motors move in discrete steps. The fundamental resolution is determined by the motor's step angle (e.g., 1.8° per full step, or smaller with microstepping). This quantization limits the smallest achievable increment of motion. For a motor with N steps per revolution and a gear reduction R, the output resolution is

$360/(N \cdot R)$ degrees. For the screw motor, the linear resolution depends on the lead screw pitch and motor step angle. *(Students to calculate and insert these resolutions for each joint based on their motor specs, gear ratios, and screw pitch.)*

- **Reduction Gear Accuracy:** The precision of the gear teeth in the reducers affects the uniformity of motion transmission and can introduce small periodic errors (kinematic errors) in the output angle.

- **Missed Steps:** While stepper motors are generally reliable in open-loop control if operated within their torque-speed envelope, there is a potential for missed steps under conditions of excessive load, rapid acceleration/deceleration, or insufficient motor torque. The design of the V2 arm, particularly the screw motor for Joint 2, aimed to provide sufficient torque to minimize this risk. *(Mention if any conditions leading to missed steps were observed or are considered a risk.)*

5.1.3 Control System Uncertainties

- **Timing Precision:** The STM32 microcontroller generates pulse trains to drive the stepper motors. The precision of these pulses, determined by the MCU's clock stability and interrupt latency, is generally very high (typically in the nanosecond to microsecond range) and unlikely to be a significant source of mechanical uncertainty compared to other factors.

- **Open-Loop Control Limitations:** The primary control loop for motor positioning is open-loop. This means the system commands a certain number of steps but does not have direct feedback on the actual achieved position of each joint for real-time correction (though the TFT display polls for status, this is not for closed-loop control of position). This makes the system susceptible to uncorrected deviations from the above-mentioned mechanical and actuation uncertainties.

- **Homing (Zeroing) Instability:** Following an emergency stop, the robotic arm's homing sequence may require manual reset. The re-established zero position can exhibit instability, leading to slight deviations in the origin of the coordinate system for subsequent programmed movements. This directly impacts the absolute accuracy of all following operations.

5.1.4 Environmental and Operational Uncertainties

- **Thermal Effects:** The vacuum stage operates within a magnetron sputtering machine, which can involve elevated temperatures (e.g., $300°C$ mentioned for an experiment). Thermal expansion and contraction of the aluminum components (coefficient of thermal expansion for aluminum $\approx 23 \times 10^{-6}$ m/(m°C)) can lead to changes in link lengths and joint alignments, potentially affecting calibration and positioning accuracy over varying operating temperatures. *(Estimate potential dimensional changes based on temperature range and component sizes if significant.)*

- **Vibrations:** Vibrations can originate from the stepper motors themselves, the operation of the sputtering machine, or external sources. These can affect the stability of the end-effector, particularly during fine positioning or when holding a static pose.

- **Workspace and Collision Risks:** For substrates or target fixtures that approach the limits of the robotic arm's defined workspace, the current system lacks advanced sensing for precise object measurement or dynamic collision avoidance. This creates a risk of collision or improper handling if objects exceed expected dimensions or are unexpectedly positioned.

- **In-Chamber Component Degradation:** Robotic arm components operating within the deposition chamber are exposed to plasma, energetic particle bombardment, and coating material overspray. The precise nature and rate of erosion, material redeposition, or chemical modification of critical surfaces (e.g., bearings, exposed motor parts if not perfectly shielded, cable insulation) are uncertain. Such degradation could affect long-term reliability, introduce particulate contamination into the vacuum environment, and alter the mechanical properties of components over time.

- **Repeatability Over Time:** Wear in mechanical components (gears, bearings, screw) over extended operational periods could gradually change backlash and friction characteristics, potentially degrading repeatability. This is a long-term consideration, exacerbated by the harsh vacuum environment.

5.2 Quantification and Impact on Performance

The cumulative effect of these uncertainties determines the overall positioning accuracy and repeatability of the robotic arm's end-effector.

- **Positioning Accuracy:** The verification requirement for the stepper motor system (Section 3.2.1.B) specifies that "the end position should be within 1mm tolerance" after a simple up-and-down program. This 1 mm value represents a target for overall system accuracy under specific test conditions. Achieving this consistently across the entire workspace, under various loads, and considering factors like homing instability, is a key performance indicator. *(Discuss how close the system is to achieving this or if further tests are needed, especially in light of homing issues.)*

- **Angular Accuracy:** For rotational joints, the angular accuracy is critical for orienting the substrate correctly. This is affected by motor step resolution, gear accuracy, backlash, and homing precision. *(Students should estimate or state target/measured angular accuracy, e.g., "Target angular accuracy is $\pm X$ degrees.")*

- **Impact on Nanocoating:** In the context of nanocoating, positional and orientational uncertainties of the substrate directly impact the uniformity, thickness, and quality of the deposited film. Inconsistent distances or angles relative to the sputtering source, or particulate contamination from degraded components, can lead to variations in coating properties and potentially compromise the functionality of the coated part.

5.3 Mitigation Strategies

Several design choices and operational procedures aim to mitigate these uncertainties:

- Robust mechanical design (V2 iteration) to improve stiffness and torque delivery (e.g., screw motor for Joint 2, selection of appropriate aluminum profiles).

- Selection of appropriate stepper motors and reducers to meet torque and resolution requirements.
- Careful assembly and alignment procedures.
- *Potential for improved homing procedures or sensors to enhance zero-point stability.*
- *Consideration of shielding or material selection for components exposed to plasma to reduce degradation.*
- Software compensation for backlash (if implemented or planned, e.g., adding extra steps after a direction reversal).
- Potential for calibration routines to map and compensate for systematic errors across the workspace. *(If calibration is planned or implemented, describe its principle here.)*
- Operating the system within its designed load and speed capacities to prevent missed steps or excessive deflections.
- *Implementation of soft limits or basic workspace monitoring to reduce collision risk.*

5.4 Conclusion on Uncertainties

The four-axis vacuum stage, like all complex mechatronic systems, is subject to a variety of uncertainties. While the design incorporates measures to minimize these, their combined effect defines the ultimate precision and repeatability. The target accuracy of within 1 mm for end-effector positioning is a critical benchmark, though achieving this reliably is challenged by factors such as homing instability and potential environmental degradation. The current prototype is a laboratory-scale system, and further uncertainties exist regarding its direct scalability for larger substrates, higher throughput, or the more demanding conditions of industrial manufacturing. Further characterization, including long-term endurance testing in the operational environment and potentially employing external metrology, is essential to fully quantify these uncertainties and their impact on the nanocoating process. This will ensure the stage can meet the stringent requirements of advanced nano-manufacturing and guide future refinements.

6 Future Work

1. Action Program: Action smoothness can be improved by using acceleration calculation. More sets of movements for substrate of different shapes and size can be designed.

2. Interface: More interactive controls on the touch screen for switching presets for different objects and adjusting parameters. More information to be shown on touch screen like time, power and so on.

3. System access: Allowing the stage to be controlled directly from the coating system or remotely from a computer.

4. Circuit: Using PCB to integrate more parts like the MCU, stepper controller and cables to save costs.

5. Mechanical: Reduce the clearance of angle switcher at the bearing location. Improved stability and precision

6. Experiment: test substrate with more different shapes.

7 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.

8 Schedule

Week	Songyuan Lyu	Xingjian Kang
2/24/25	Define the basic dimensions and geometric structure of the robotic arm as a foundation for kinematic analysis.	Compare potential motor models and match them with suitable motor controllers.
3/3/25	Establish coordinate frames and kinematic equations for the robotic arm. Set up the initial MATLAB code structure.	Pick an MCU with sufficient performance to handle communication and control tasks.
3/10/25	Further refine the forward and inverse kinematics in MATLAB, and develop the trajectory planning code. Apply the Jacobian method to determine joint angles.	Develop the system schematic, detailing power distribution, signal lines, motor driver connections, and any required safety features.
3/17/25	Use MATLAB to plot joint angles and create animation of the iteration process. Verify the correctness and stability of the kinematic algorithms.	Install Keil IDE, set up the toolchain, and explore sample projects. Familiarize with STM32's hardware abstraction layer (HAL) or low-level (LL) libraries.
3/24/25	Calculate required torque and speed based on the expected load of CNC aluminum components. Finalize stepper motor specifications and determine an appropriate gear reduction ratio.	Conduct fundamental tests (LED blinking, reading sensor inputs). Make sure the development board work correctly.
3/31/25	Write control software to drive stepper motors and achieve precise joint control. Perform testing and fine-tuning to align hardware performance with MATLAB simulations.	Test RS485 from STM32F407 development board to "Emm42" stepper controller
4/7/25	Combine hardware, software, and control algorithms for a preliminary system test. Test movement range, joint actions, and error margins, recording data for further improvements.	Test TFT touchscreen functionality. Visualize control parameters.
4/14/25	Identify improvements based on the first-generation test results. Design and sketch the second-generation arm structure, focusing on enhanced stability using a jack mechanism.	Build a simple interface on the STM32 screen (or external display) showing motor positions, speeds, errors, etc.
4/21/25	Develop kinematic and dynamic models for the revised structure. Assess load, inertia, and coupling between joints.	Build a set of actions for arm motion. Assign functions to physical buttons and touchscreen virtual buttons.
4/28/25	Visualize the new arm's trajectory in a simulation environment, validating joint angles, velocities, and accelerations. Refine control strategies to minimize vibration and impact.	Optimize motion profiles and timing to ensure smooth, consistent coating.

Week	Songyuan Lyu	Xingjian Kang
5/5/25	Verify component specifications and machining tolerances. Begin assembling the second-generation robotic arm.	Verify that all four motors can start, stop, and coordinate movements in unison.
5/12/25	Complete the mechanical, electronic, and control system integration. Test motion performance and communication latency in a real hardware environment.	Use the kinematic algorithms to command precise motion of all four motors.
5/19/25	Establish communication with the host computer or other devices to enable real-time monitoring and control. Optimize the protocol and data handling to reduce transmission delays.	Adjust the system based on test feedback (reduce vibrations, enhance control accuracy).
5/26/25	Conduct comprehensive performance evaluations, including accuracy, speed, and stability. Collect data and make any last-minute refinements to the algorithms or hardware design. Prepare the final presentation.Prepare the final presentation.	Thoroughly test simultaneous multi-motor motion, communication stability, and user interface.Prepare the final presentation.

Week	Yanjie Li	Yanghonghui Chen
2/24/25	Continue creating initial component models (e.g., motors, linkages). Verify dimensional feasibility and alignment within the overall design.	Confirm voltage/current needs and RS485 communication requirements for each motor controller.Identify required safety features (e.g., overvoltage protection, filtering).
3/3/25	Use FEA to determine the required arm length for achieving coating objectives. Evaluate stresses and deformations under expected loads.	Finalize the plan to place four RS485 module boards on a single PCB.Determine the physical arrangement and spacing to minimize interference.
3/10/25	Finalize 3D models for motors, linkage components, aluminum profiles, and coating environment. Check assembly compatibility in Fusion 360.	Plan a regulated 24V input shared by four independent power branches. Include filtering (bypass capacitors, ferrite beads) and protection (TVS diodes) for each branch.
3/17/25	Fabricate the initial robotic arm parts using 3D printing. Prepare for mechanical assembly by ensuring print quality and dimensional accuracy.	Lay out all connections (24V, GND, RS485 lines) in a schematic tool. Specify components (connectors, diodes, capacitors, etc.) and create a bill of materials.
3/24/25	Assemble the printed mechanical components. Work with the electronics team to integrate motor control hardware and ensure compatibility.	Position the four RS485 modules to minimize thermal and signal interference. Ensure proper trace width and spacing for high-current paths and differential signals.

Week	Yanjie Li	Yanghonghui Chen
3/31/25	Run basic functional tests (e.g., motion range, joint stability) outside the coating machine. Assess load-bearing performance to confirm it meets initial design parameters.	Link the eight A/B lines (four pairs) in parallel on the PCB. Implement termination and biasing resistors as needed for the single differential bus.
4/7/25	Move the prototype into the coating machine for preliminary, careful testing. Verify that the arm can operate safely and reach necessary positions for coating.	Send the finalized design files to a PCB manufacturer. Assemble the board, mount the RS485 modules, connectors, and essential components.
4/14/25	Collect feedback from in-chamber tests. Adjust arm geometry, material choices, or fastening methods to improve reliability.	Verify each branch's voltage output, current handling, and protection features. Confirm continuity and proper routing of RS485 signals.
4/21/25	Order aluminum profiles and any additional custom parts for the revised design. Confirm delivery timelines for final assembly.	Connect the PCB's A/B outputs to the STM32's RS485 transceiver. Test basic data transmission to a single motor controller to confirm signal integrity.
4/28/25	Replace or enhance 3D-printed parts with machined or aluminum-profile components as needed. Validate the assembled structure for final shape, strength, and functionality.	Power all four motor controllers simultaneously. Validate reliable half-duplex data exchange with all four boards using the STM32's DE/RE pins.
5/5/25	Conduct formal coating trials in the actual production environment. Gather data on coating uniformity, repeatability, and throughput.	Check for signal interference or crosstalk among the four modules. Introduce additional protection (ferrite beads, improved grounding, etc.) if needed.
5/12/25	Refine control strategies, motion profiles, and mechanical alignments based on test results. Make any small design tweaks for improved performance or ease of maintenance.	Combine the PCB with the robotic arm's mechanical and control systems. Test synchronized motor movement across all four axes, ensuring precise and stable motion.
5/19/25	Run the arm for longer durations to confirm long-term stability. Identify potential failure points and address them before final demonstration.	Adjust any resistor values or PCB design factors if voltage drops or noise occur under load. Validate the overall reliability and stability during extended operation.
5/26/25	Showcase the finished robotic arm performing coating tasks under real operating conditions. Present documentation of the design process, test data, and outcomes (e.g., FEA, load tests). Prepare the final presentation.	Compile final PCB documentation, including schematics, layout files, and component details. Transition to full deployment and conduct long-duration stress tests to verify the system's robustness.Prepare the final presentation.

9 Ethics

As members of the engineering and scientific community, we recognize the importance of adhering to the IEEE Code of Ethics in the design, development, and implementation of our project.

9.1 Upholding Integrity, Responsibility, and Ethical Conduct

1. Safety, Health, and Welfare of the Public

   The primary aim of this project is to enhance the quality of nano-coating processes, which have wide-ranging industrial and societal applications. By improving coating uniformity and mechanical properties, the project contributes to the development of safer and more reliable nanotechnology-based products. We will ensure that our design complies with ethical engineering practices and sustainable development principles to minimize waste and environmental impact.

2. Transparency in Risks and Limitations

   We will disclose any limitations or potential risks associated with the vacuum stage, such as mechanical failures or environmental hazards from the magnetron sputtering process. These disclosures will help stakeholders make informed decisions about the deployment of the technology.

3. Avoiding Conflicts of Interest

   We will avoid any real or perceived conflicts of interest in our professional activities. If such conflicts arise, we will disclose them to the relevant parties to ensure transparency and maintain trust.

4. Rejection of Unlawful Conduct and Bribery

   Throughout the project, we commit to adhering to all legal and ethical standards. We reject any form of bribery or unethical practices in procurement, manufacturing, or collaboration processes.

5. Acknowledgment and Correction of Errors

   We will seek and accept honest criticism of our work, acknowledging and correcting any errors identified during the design, testing, or implementation phases. Proper credit will be given to all contributors, including team members and external collaborators, for their intellectual and technical contributions.

6. Competence and Qualification

   We commit to undertaking tasks for which we are qualified through training and experience. If limitations in our expertise arise, we will seek additional guidance, training, or collaboration to ensure the project's success.

9.2 Treating All Persons Fairly and Respectfully

Our team is committed to treating all members, collaborators, and stakeholders with fairness and respect. We will not engage in discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression. We will foster a respectful and inclusive working environment, ensuring that no individual is subjected to harassment, bullying, or any form of inappropriate behavior. The design of the robotic arm will prioritize the safety of operators and users. We will avoid false or malicious actions that could harm others' property, reputation, or employment. All safety measures will be thoroughly tested and documented to prevent injury or damage during operation.

9.3 Ensuring Ethical Conduct Among Colleagues

We will actively encourage our team members and collaborators to adhere to the IEEE Code of Ethics. Through regular meetings and discussions, we will ensure that ethical considerations are integrated into every stage of the project. We will report and address any ethical violations that may arise during the project. Retaliation against individuals who report violations will not be tolerated.[9]

10 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.

[9] "Ieee code of ethics," Website, https://www.zotero.org/user/validate/qz80L672HXcHeg5cuBSinzvaLKKZO3.

Appendix A - Main Program

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

// --- State Machine Definition ---
enum MoveState1 {
    MS1_IDLE = 0,
    MS1_STEP1,  // Initial move
    MS1_WAIT1,  // Wait for 30s
    MS1_STEP2,  // Accelerate
    MS1_WAIT2,  // Wait for 60s
    MS1_STEP3,  // Decelerate and return to initial
    MS1_WAIT3,  // Wait for the motors to stop
    MS1_DONE    // Done, can be reset
};

enum MoveState2 {
    MS2_IDLE = 0,
    MS2_STEP1,  // Initial move
    MS2_WAIT1,  // Wait for 30s
    MS2_STEP2,  // Accelerate
    MS2_WAIT2,  // Wait for 60s
    MS2_STEP3,  // Decelerate and return to initial
    MS2_WAIT3,  // Wait for the motors to stop
    MS2_DONE    // Done, can be reset
};

static MoveState1 move1_state = MS1_IDLE;
static MoveState2 move2_state = MS2_IDLE;
static uint32_t move1_lastTick = 0;

const char* MoveState1Names[] = {
    "IDLE",
    "STEP1",
    "WAIT1",
    "STEP2",
    "WAIT2",
    "STEP3",
    "WAIT3",
    "DONE"
};

// Define the message display area size
#define MSG_X    30
#define MSG_Y    150
#define MSG_W    500
```

```
#define MSG_H    100
// Define a unified output function that prints with fixed width and pads spaces automatically
void displayMessage(const char* msg) {
    // Assume the message area occupies a total of 20 characters in width
    char buf[32];
    snprintf(buf, sizeof(buf), "%-20s", msg);
    // "%-20s": left-aligned, total of 20 characters, padded with spaces

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


class Motor {
private:
    uint8_t addr;          // Motor address
    uint8_t set_dir;       // Set forward direction (0 or 1)
    uint8_t dir;           // Current direction bit sent to the motor
    uint8_t redu_ratio;    // Reduction ratio of the motor gearbox
    uint8_t acc;           // Acceleration parameter for the motor
    uint16_t vel;          // Target velocity (raw value to be sent to motor)

public:
    int velocity;          // User-defined speed (RPM)
    double tgt_degree;     // Target angle (degrees)
    int32_t read_velocity; // Real-time read velocity (RPM)
    int32_t read_position_raw;// Real-time read raw position count (signed)
    double read_degree;    // Real-time read angle (degrees)
    bool reach_pos;
    int duration; // Duration for the motor to reach the target position
    char* status; // Status string for the motor

    // Initializes the motor address, direction, reduction ratio, and acceleration
    void init(
        uint8_t address, uint8_t direction = 0, uint8_t reduction_ratio = 1, uint8_t acc_val = 10
    ) {
        addr = address;
        set_dir = direction;
        redu_ratio = reduction_ratio;
        acc = acc_val;
        velocity = 0;
        tgt_degree = 0;
        read_velocity = 0;
```

```
        read_position_raw = 0;
        read_degree = 0;
        status = "STOP";

    }

    // Sets the target position (in degrees) and optionally the velocity, then sends the position control command
    uint32_t tgt_position(double degree, uint16_t velocity_val = 1) { // [/3200 = round] - Comment likely referring to a rounding aspect in the underlying implementation
        tgt_degree = degree;
        read_degree += degree; // Update the real-time read angle
        if ((velocity_val == 1 && vel == 0) || (velocity_val != 1)) {
            vel = velocity_val; // [RPM] - Set velocity if a velocity value is provided
        }
        if (degree < 0) {
            dir = set_dir ? 0 : 1; // Reverse direction if the degree is negative
            degree = -degree; // Store the absolute value of the degree
            velocity = -vel;
        } else {
            dir = set_dir;
            velocity = vel;
        }
        uint32_t position = degree * 3200 * redu_ratio /360; // [/3200 = 1 round] - Comment likely referring to a rounding aspect in the underlying implementation
        duration = degree * redu_ratio *60000 /360 / vel +1000; //[ms] // Calculate the duration to reach the target position

        Emm_V5_Pos_Control(addr, dir, vel, acc, position, 0, 0); // [degree] - Send position control command to the motor
        return position;
    }

    // Sets the velocity and direction based on the input velocity value, without sending a command
    void set_velocity(int velocity_val) {
        velocity = velocity_val;
        if (velocity < 0) {
            vel = static_cast<uint16_t>(-velocity); // Store the absolute value of the negative velocity
            dir = set_dir ? 0 : 1;             // Reverse direction if the velocity is negative
        } else {
```

```
            vel = static_cast<uint16_t>(velocity); // Store the positive
                velocity
            dir = set_dir;                         // Maintain the set
                forward direction
        }
    }

    // Sends the constant speed control command to the motor
    void constant_rorate() {
        Emm_V5_Vel_Control(addr, dir, vel, acc, 0);
    }

    void constant_rorate(int velocity_val) {
        if(velocity_val == 0) {
            status = "STOP";
        }
        set_velocity(velocity_val);
        Emm_V5_Vel_Control(addr, dir, vel, acc, 0);
    }

    // Returns the reduction ratio of the motor
    uint8_t get_reduction_ratio() const { return redu_ratio; }

    // Displays the current status (speed, target angle, working state)
        on the TFT screen
    void displayStatus(int baseX, int baseY) const {
        char buf[32];
        const int offsetX = 10; // Horizontal spacing
        int currentY = baseY;
        int currentX = baseX;

        // --- First line: Motor label and status (STOP/RUN) ---
        // Output the label and state with fixed width
        snprintf(buf, sizeof(buf), "M%02X:", addr);
        lcd_show_string(currentX, currentY, 60, 16, 16, buf, BLUE);
        currentX += 60 + offsetX * 2;

        // const char* status = (velocity != 0) ? "RUN" : "STOP";
        // uint16_t color = (velocity != 0) ? GREEN : RED;
        // // Fixed width of 4 characters, padded with spaces on the
            right
        // snprintf(buf, sizeof(buf), "%-4s", status);
        // lcd_show_string(currentX, currentY, 60, 16, 16, buf, color);

        //status
        uint16_t color = (status[0] == 'R') ? GREEN : RED;
        snprintf(buf, sizeof(buf), "%-4s", status);
        lcd_show_string(currentX, currentY, 60, 16, 16, buf, color);
```

```
        // --- Second line (continued): SPEED and DIRECTION or STOP "0" ---
        currentY += 20;
        currentX = baseX;
        lcd_show_string(currentX, currentY, 60, 16, 16, "SPEED:", BLUE);
        currentX += 60 + offsetX;

        char displayBuf[10];
        if (velocity > 0) {
            const char* dirStr = (set_dir == 0) ? "CLW" : "CCLW";
            // Total fixed length of 8 characters: 4 for direction + 3 for speed + 1 for trailing space
            snprintf(displayBuf, sizeof(displayBuf), "%-4s%3d␣", dirStr, velocity);
        } else if (velocity < 0) {
            const char* dirStr = (set_dir == 0) ? "CCLW" : "CLW";
            snprintf(displayBuf, sizeof(displayBuf), "%-4s%3d␣", dirStr, -velocity);
        } else {
            // Speed is 0: fixed 8 characters, with number right aligned
            snprintf(displayBuf, sizeof(displayBuf), "␣␣␣␣0␣␣␣");
        }
        // Directly write on screen; trailing spaces overwrite old characters
        lcd_show_string(currentX, currentY, 70, 16, 16, displayBuf, BLUE);

        // --- Third line: TARGET DEGREE ---
        currentY += 20;
        currentX = baseX;
        lcd_show_string(currentX, currentY, 100, 16, 16, "TARGET:", BLUE);
        currentX += 100 + offsetX;
        // Target angle printed with fixed width of 5 digits (including sign)
        int tgt_int = static_cast<int>(tgt_degree);
        snprintf(buf, sizeof(buf), "%5d", tgt_int);
        lcd_show_string(currentX, currentY, 80, 16, 16, buf, BLUE);
    }
};

// Global definitions for four motor instances
Motor motor[5];

// Track the status for four motors
static uint32_t motorStartTick[4] = {0, 0, 0, 0}; // Each motor's movement start tick (HAL_GetTick())
static bool     motorMoving[4]    = {false, false, false, false}; // Indicates if the motor is in motion
```

```
int wait_time[5] = {0,0,0,0,0}; //[s] // Wait time for each motor

// void pollMotorStops() {
//     uint32_t now = HAL_GetTick();
//     for (int i = 0; i < 4; i++) {
//         if (motorMoving[i] && now - motorStartTick[i] >= motorDuration[i]) {
//             // Stop the corresponding motor
//             motor[i].constant_rorate(0);
//             motorMoving[i] = false; // Clear the movement flag
//         }
//     }
// }

// 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 = &motor[0]; break;
        case 2: pm = &motor[1]; break;
        case 3: pm = &motor[2]; break;
        case 4: pm = &motor[3]; 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;
                uint8_t rr = pm->get_reduction_ratio();
                pm->read_degree = pm->read_position_raw * 360.0 / (3200.0 * rr);
            }
            break;
        case 0x3A: // Status flag
            if (len >= 4) {
                uint8_t status = rs485buf[2];
                pm->reach_pos = (status & 0x02) ? true : false; // Check if the target position is reached
            }
            break;
        default:
            break;
    }

    // The four motors are arranged vertically, each refreshed individually
    int baseX = 30;
    for (uint8_t i = 0; i < 4; i++) {
        int y = 210 + (i + 1) * 100;
        motor[i].displayStatus(baseX, y);
    }
}

// --- Non-blocking progression function ---
void process_move_set_1(void) {
    uint32_t now = HAL_GetTick();
    switch (move1_state) {
        case MS1_IDLE:
            // Idle - waiting for trigger
            break;

        case MS1_STEP1:
            // Step 1: All four motors act simultaneously
            motor[0].tgt_position(18, 40); //[degree]
            motorStartTick[0] = HAL_GetTick();
            motorMoving[0]   = true;
            delay_ms(10);
            motor[1].tgt_position(40, 30); //[mm]
            motorStartTick[1] = HAL_GetTick();
            motorMoving[1]   = true;
            delay_ms(10);
            motor[2].tgt_position(0, 10); //[degree]
            motorStartTick[2] = HAL_GetTick();
            motorMoving[2]   = true;
```

```
            delay_ms(10);
            motor[3].constant_rorate(3);
            delay_ms(10);
            for(int i = 0; i < 4; i++) {
                motor[i].status = "RUN";
            }

            for(int i = 0; i < 3; i++)
                if(motor[i].duration > wait_time[0]*1000) {
                    wait_time[0] = motor[i].duration/1000;
                }

            move1_lastTick = now;
            move1_state = MS1_WAIT1;
            break;

        case MS1_WAIT1:
            // Wait for 30 seconds (30000 ms)
            for(int i = 0; i < 3; i++) {
                if (now - motorStartTick[i] >= motor[i].duration) {
                    motor[i].status = "STOP";
                    motor[i].duration = 0;
                }
            }
            if (now - move1_lastTick >= wait_time[0]*1000) {
                move1_state = MS1_STEP2;
            }
            break;

        case MS1_STEP2:
            // Step 2: Raise the platform
            motor[0].tgt_position(-5, 5);
            motorStartTick[0] = HAL_GetTick();
            motorMoving[0]   = true;
            delay_ms(10);
            motor[1].tgt_position(-10, 2);
            motorStartTick[1] = HAL_GetTick();
            motorMoving[1]   = true;
            delay_ms(10);
            motor[2].tgt_position(30, 5);
            motorStartTick[2] = HAL_GetTick();
            motorMoving[2]   = true;
            delay_ms(10);
            for(int i = 0; i < 4; i++) {
                motor[i].status = "RUN";
            }
            for(int i = 0; i < 3; i++)
                if(motor[i].duration > wait_time[1]*1000) {
```

```
                wait_time[1] = motor[i].duration/1000;
            }

        move1_lastTick = now;
        move1_state = MS1_WAIT2;
        break;

    case MS1_WAIT2:
        // Wait another 60 seconds (60000 ms)
        for(int i = 0; i < 3; i++) {
            if (now - motorStartTick[i] >= motor[i].duration) {
                motor[i].status = "STOP";
                motor[i].duration = 0;
            }
        }
        if (now - move1_lastTick >= wait_time[1]*1000) {
            move1_state = MS1_STEP3;
        }

        break;

    case MS1_STEP3:
        // Step 3: Return to home
        motor[0].tgt_position(-13, 40);
        motorStartTick[0] = HAL_GetTick();
        motorMoving[0]   = true;
        delay_ms(10);
        motor[1].tgt_position(-30, 8);
        motorStartTick[1] = HAL_GetTick();
        motorMoving[1]   = true;
        delay_ms(10);
        motor[2].tgt_position(-30, 3);
        motorStartTick[2] = HAL_GetTick();
        motorMoving[2]   = true;
        delay_ms(10);
        motor[3].constant_rorate(0);
        delay_ms(10);
                            for(int i = 0; i < 3; i++) {
            motor[i].status = "RUN";
        }

        move1_lastTick = now;
        wait_time[2] = 0; // Reset wait time for the next step
        for(int i = 0; i < 3; i++)
            if(motor[i].duration > wait_time[2]*1000) {
                wait_time[2] = motor[i].duration/1000;
            }
        wait_time[2] += 5; // Add a 5-second buffer to the wait time
```

```
            motor[3].status = "STOP";
            move1_state = MS1_WAIT3;
            break;

        case MS1_WAIT3:
            // Wait for the motors to stop
            for(int i = 0; i < 3; i++) {
                if (now - motorStartTick[i] >= motor[i].duration) {
                    motor[i].status = "STOP";
                    motor[i].duration = 0;
                }
            }
            if (now - move1_lastTick >= wait_time[2]*1000) {
                move1_state = MS1_DONE;
            }
            break;

        case MS1_DONE:
            // After completion, either automatically return to IDLE or
               remain DONE until Key0 re-triggers
            for(int i = 0; i < 4; i++) {
                motor[i].status = "STOP";
            }
            move1_state = MS1_IDLE;
            break;
    }
}

void process_move_set_2(void) {
    uint32_t now = HAL_GetTick();
    switch (move2_state) {
        case MS2_IDLE:
            // Idle - waiting for trigger
            break;

        case MS2_STEP1:
            // Step 1: All four motors act simultaneously
            motor[0].tgt_position(18, 50); //[degree]
            motorStartTick[0] = HAL_GetTick();
            motorMoving[0]   = true;
            delay_ms(10);
            motor[1].tgt_position(40, 40); //[mm]
            motorStartTick[1] = HAL_GetTick();
            motorMoving[1]   = true;
            delay_ms(10);
            motor[2].tgt_position(5, 15); //[degree]
            motorStartTick[2] = HAL_GetTick();
            motorMoving[2]   = true;
```

```
            delay_ms(10);
            motor[3].constant_rorate(50);
            delay_ms(10);
            for(int i = 0; i < 4; i++) {
                motor[i].status = "RUN";
            }

            for(int i = 0; i < 3; i++)
                if(motor[i].duration > wait_time[0]*1000) {
                    wait_time[0] = motor[i].duration/1000;
                }

            move1_lastTick = now;
            move2_state = MS2_WAIT1;
            break;

        case MS2_WAIT1:
            // Wait for 30 seconds (30000 ms)
            for(int i = 0; i < 3; i++) {
                if (now - motorStartTick[i] >= motor[i].duration) {
                    motor[i].status = "STOP";
                    motor[i].duration = 0;
                }
            }
            if (now - move1_lastTick >= wait_time[0]*1000) {
                move2_state = MS2_STEP2;
            }
            break;

        case MS2_STEP2:
            // Step 2: Raise the platform
            motor[0].tgt_position(-5, 10);
            motorStartTick[0] = HAL_GetTick();
            motorMoving[0]   = true;
            delay_ms(10);
            motor[1].tgt_position(-40, 5);
            motorStartTick[1] = HAL_GetTick();
            motorMoving[1]   = true;
            delay_ms(10);
            motor[2].tgt_position(50, 5);
            motorStartTick[2] = HAL_GetTick();
            motorMoving[2]   = true;
            delay_ms(10);
            for(int i = 0; i < 4; i++) {
                motor[i].status = "RUN";
            }
            for(int i = 0; i < 3; i++)
                if(motor[i].duration > wait_time[1]*1000) {
```

```
                wait_time[1] = motor[i].duration/1000;
            }

        move1_lastTick = now;
        move2_state = MS2_WAIT2;
        break;

    case MS2_WAIT2:
        // Wait another 60 seconds (60000 ms)
        for(int i = 0; i < 3; i++) {
            if (now - motorStartTick[i] >= motor[i].duration) {
                motor[i].status = "STOP";
                motor[i].duration = 0;
            }
        }
        if (now - move1_lastTick >= wait_time[1]*1000) {
            move2_state = MS2_STEP3;
        }

        break;

    case MS2_STEP3:
        // Step 3: Return to home
        motor[0].tgt_position(-13, 50);
        motorStartTick[0] = HAL_GetTick();
        motorMoving[0]   = true;
        delay_ms(10);
        motor[1].tgt_position(0, 15);
        motorStartTick[1] = HAL_GetTick();
        motorMoving[1]   = true;
        delay_ms(10);
        motor[2].tgt_position(-55, 15);
        motorStartTick[2] = HAL_GetTick();
        motorMoving[2]   = true;
        delay_ms(10);
        motor[3].constant_rorate(0);
        delay_ms(10);
                    for(int i = 0; i < 3; i++) {
            motor[i].status = "RUN";
        }

        move1_lastTick = now;
        wait_time[2] = 0; // Reset wait time for the next step
        for(int i = 0; i < 3; i++)
            if(motor[i].duration > wait_time[2]*1000) {
                wait_time[2] = motor[i].duration/1000;
            }
        wait_time[2] += 5; // Add a 5-second buffer to the wait time
```

```c
            motor[3].status = "STOP";
            move2_state = MS2_WAIT3;
            break;

        case MS2_WAIT3:
            // Wait for the motors to stop
            for(int i = 0; i < 3; i++) {
                if (now - motorStartTick[i] >= motor[i].duration) {
                    motor[i].status = "STOP";
                    motor[i].duration = 0;
                }
            }
            if (now - move1_lastTick >= wait_time[2]*1000) {
                move2_state = MS2_DONE;
            }
            break;

        case MS2_DONE:
            // After completion, either automatically return to IDLE or
                remain DONE until Key0 re-triggers
            for(int i = 0; i < 4; i++) {
                motor[i].status = "STOP";
            }
            move2_state = MS2_IDLE;
            break;
    }
}

void manual_set_1() {
    motor[0].tgt_position(-20, 30);
    delay_ms(10);
    motor[1].tgt_position(-15, 10);
    delay_ms(10);
    motor[2].tgt_position(0, 20);
    delay_ms(10);
    // motor[3].tgt_position(20,10);
}

int main(void) {
    uint8_t key, t = 0, cnt = 0;
    uint8_t rs485buf[8];
    // char stateBuf[16];   // Declare buffer here

    HAL_Init();
    sys_stm32_clock_init(336, 8, 2, 7);
    delay_init(168);
    usart_init(115200);
    led_init();
```

```
    lcd_init();
    key_init();
    rs485_init(115200);
          wait_time[0] = 5*60; //[s]
    wait_time[1] = 30*60;

    // x, y, w, h, font_size
    lcd_show_string(
        30, 50, 200, 16, 16,
        "Senior Design:", RED
    );
    lcd_show_string(
        30, 70, 320, 16, 16,
        "Four-Axis Vacuum Stage", RED
    );
    lcd_show_string(
        30, 90, 320, 16, 16,
        "for Advanced Nano-Manufacturing", RED
    );

    // Motor initialization
    // addr, dir, redu_ratio, acc
    motor[0].init(1, 1, 100, 128);
    motor[1].init(2, 1, 30, 128);
    motor[2].init(3, 0, 10, 64);
    motor[3].init(4, 0, 1, 1);

    while (1) {
        key = key_scan(0);

        if (key == KEY0_PRES && move1_state == MS1_IDLE) {
            displayMessage("Running Main Program");
            wait_time[0] = 5*60; //[s]
            wait_time[1] = 30*60;
            move1_state = MS1_STEP1;    // Trigger state machine step 1
        }
        // if (key == KEY0_PRES && move2_state == MS2_IDLE) {
        //     displayMessage("Running Demo Program");
        //     wait_time[0] = 20; //[s]
        //     wait_time[1] = 20;
        //     move2_state = MS2_STEP1;    // Trigger state machine step 1
        // }
        if (key == KEY1_PRES) {
            displayMessage("Manual Control");
            manual_set_1();
        }
        if (key == KEY2_PRES) {
            displayMessage("STOP");
```

```
            // Stop all motors
            motor[0].tgt_position(0, 0);
            delay_ms(10);
            motor[1].tgt_position(0, 0);
            delay_ms(10);
            motor[2].tgt_position(0, 0);
            delay_ms(10);
            motor[3].tgt_position(0, 0);

            move1_state = MS1_IDLE; // Interrupt state machine
            move2_state = MS2_IDLE; // Interrupt state machine
        }

        // show wait_time
        lcd_show_string(30, 170, 320, 16, 16, "step1 time:", BLUE);
        lcd_show_num(120, 170, wait_time[0], 6, 16, BLUE);
        lcd_show_string(30, 190, 320, 16, 16, "step2 time:", BLUE);
        lcd_show_num(120, 190, wait_time[1], 6, 16, BLUE);

        process_move_set_1();        // Drive the state machine (non-blocking)
        process_move_set_2();
        // pollMotorStops();           // Auto-stop check for motors

        // Assume the maximum status name length of 6 characters, such as "WAIT2" or "STEP1"
        // Format it to fixed length 6 with a prefix; this ensures that if the status is "IDLE" (4 characters),
        // two trailing spaces are added to maintain consistent display width
        char stateBuf[16];
        if(move1_state == MS1_IDLE && move2_state != MS2_IDLE) {
            snprintf(stateBuf, sizeof(stateBuf), "MS2:%-6s", MoveState1Names[(int)move2_state]);
        } else {
            snprintf(stateBuf, sizeof(stateBuf), "MS1:%-6s", MoveState1Names[(int)move1_state]);
        }
        //snprintf(stateBuf, sizeof(stateBuf), "MS1:%-6s", MoveState1Names[(int)move1_state]);
        lcd_show_string(30, 230, 200, 16, 16, stateBuf, BLUE);

        // Periodically read RS485 and update the display
        if (++t >= 20) {
            t = 0;
            LED0_TOGGLE();
            cnt++;
            lcd_show_xnum(78, 130, cnt, 3, 16, 0x80, BLUE);
```

```c

            for (uint8_t addr = 1; addr <= 4; ++addr) {
                Emm_V5_Read_Sys_Params(addr, S_VEL);
                Emm_V5_Read_Sys_Params(addr, S_CPOS);
                Emm_V5_Read_Sys_Params(addr, S_FLAG);
            }
        }

        Translate_received_data(rs485buf);

        HAL_Delay(1); // a very short delay to maintain the main loop
            frequency while remaining non-blocking
    }

    return 0;
}
```

Appendix B - Engineering Drawing

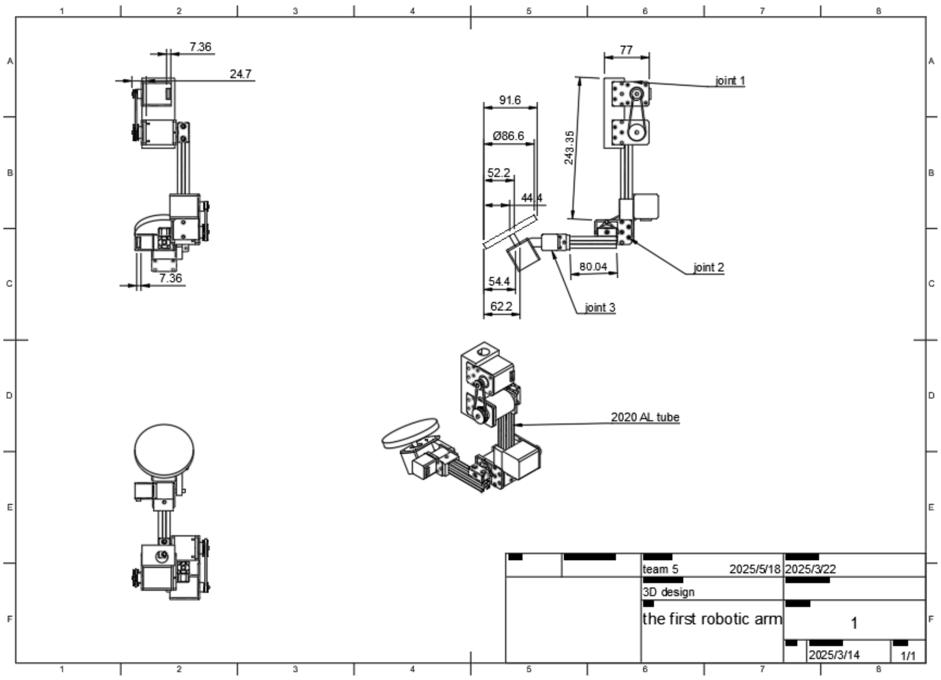


Figure 22: The Engineering Drawing of version one

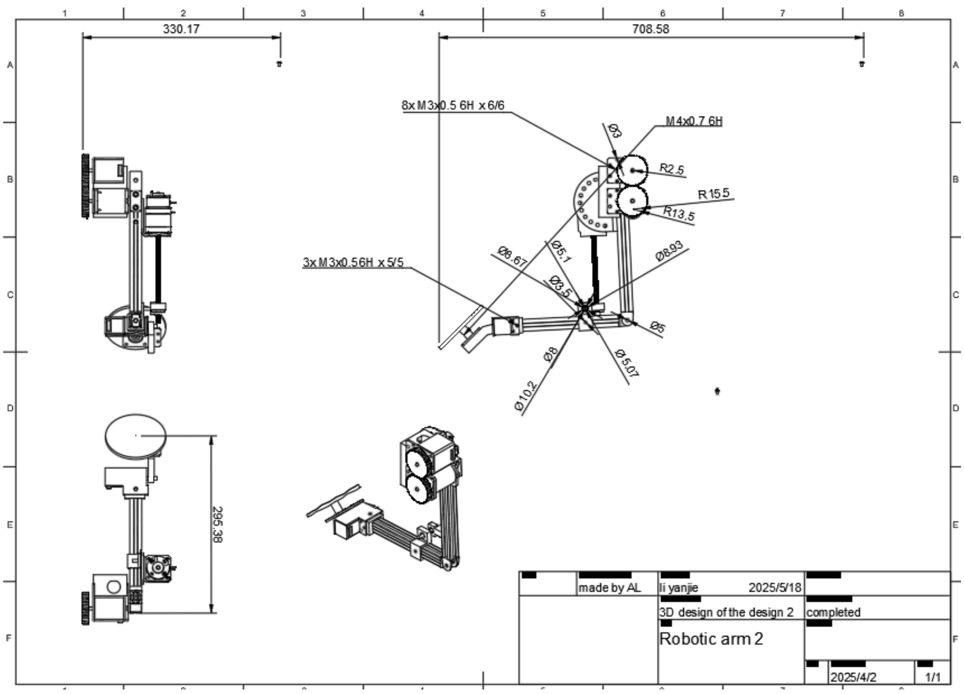


Figure 23: The Engineering Drawing of version one