ECE 445

Senior Design Laboratory

Final Report

Human-Robot Interaction for Object Grasping with Visual Reality and Robotic Arms

---

Team #42

Ziming Yan (zimingy3@illinois.edu)

Jiayu Zhou (jiayu9@illinois.edu)

Yuchen Yang (yucheny8@illinois.edu)

Jingxing Hu (hu80@illinois.edu)

Advisor: Prof. Liangjing Yang &

Prof. Gaoang Wang

TA: Tielong Cai & Tianci Tang

May 19, 2025

Abstract

We present a VR-guided robotic claw system that enables intuitive remote manipulation through a Unity-based digital twin and Meta Quest VR interface. This system aims at assistive and industrial applications, it allows users to grasp objects in real time using a 3D virtual replica synchronized with a UR3e robotic arm. The control module incorporates an STM32-based closed-loop force control system, leveraging UART and CAN communication to drive a 42 stepper motor. The digital twin provides high-fidelity simulation at over 250 FPS and real-time feedback, while the VR interface offers immersive control with sub-50 ms latency. Our design demonstrates reliable performance in both functional grasping tasks and remote operation scenarios.

Contents

1 Introduction

This project addresses the need for natural and efficient human-robot interaction in scenarios where direct manipulation is impractical. Our system enables users to control a robotic arm in real time through a Virtual Reality (VR) interface, allowing for intuitive object grasping from a distance. The primary motivation stems from assistive care: for example, enabling hospital patients confined to bed to grasp nearby objects on a table without physical effort. Beyond assistive use, the system also holds potential for remote industrial manipulation in constrained or hazardous environments—such as during a pandemic—where human presence is limited. By combining Meta Quest VR [1], Unity-based digital twins [2], [3], and the UR3e robotic arm, we bridge human intent and robotic execution with high precision, low latency, and broad applicability.

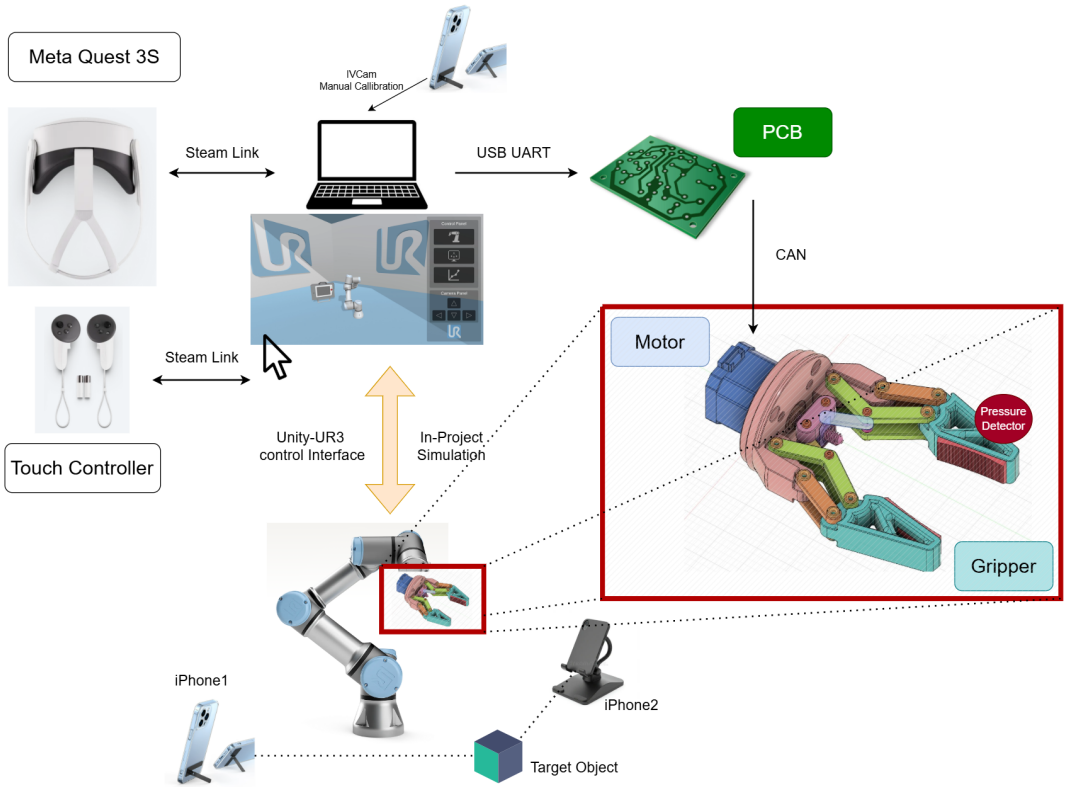

Figure 1: System Workflow: VR-based control of UR3e robotic arm using Meta Quest 3S, Unity interface, and IVCam-assisted manual calibration.

2 Design

2.1 Design description

Generally, we developed the system which we used to achieve our senior design project, and we drew the block diagram to delineate the subsystems and individual tasks, which is shown as Figure 2.The system is composed of four main modules: Control Module, Actuator Module, User Interaction Module, and Power Module, collaboratively enabling a VR-guided robotic claw system with closed-loop force control and CAN communication.

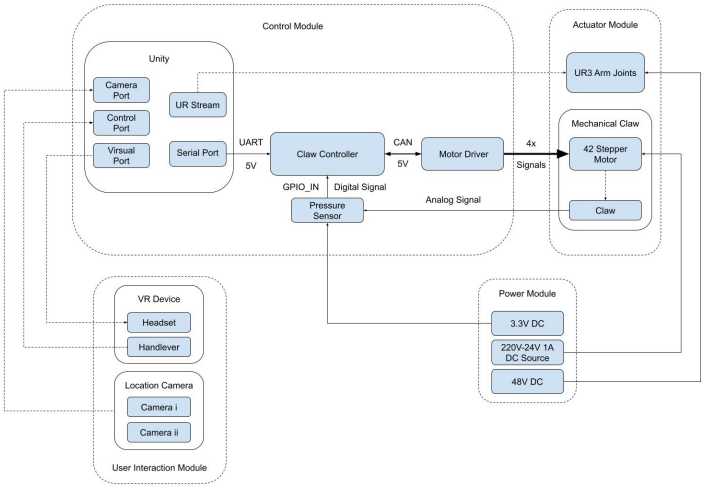

Figure 2: Block Diagram

Control Module

At the core of the system lies the Control Module, where Unity software runs and interfaces with the physical system via multiple communication ports. The UR Stream and Serial Port in Unity handle command and signal exchange through UART with the Claw Controller (based on an STM32 microcontroller). The controller processes incoming signals and sends control commands via CAN bus to the Motor Driver, which in turn drives the 42 Stepper Motor. Additionally, a Pressure Sensor provides real-time feedback to the Claw Controller via analog and GPIO inputs, forming a basic closed-loop control system for grasping force regulation.

Digital Twin

In Unity, we create the digital twin of the UR3e robot arm and our self-designed claw. We provide the control interface as well as the Real-time 3D position for the user to locate the state of the robot arm.

Actuator Module

The Actuator Module consists of a 42 Stepper Motor and a 3D-printed claw. The motor is controlled by the Motor Driver, which receives commands from the Claw Controller via CAN bus. The claw's design allows for precise control of the grasping force, enabling the system to adapt to various objects and scenarios.

User Interaction Module

The User Interaction Module is designed to facilitate user engagement with the system. It includes a VR headset and controllers, which provide an immersive experience for the user. The Unity software processes input from the VR controllers and translates it into commands for the Claw Controller, allowing users to interact with virtual objects in a realistic manner.

Power Module

The Power Module supplies the necessary power to the entire system. It ensures that all components, including the Control Module, Actuator Module, and User Interaction Module, receive stable and sufficient power for optimal performance.

Initially, we planned to use a STM32F103 microcontroller and SG-90 motor to control the claw. However, we found the accuracy was low and reaction time was far from satisfactory. Therefore, we changed to use more powerful STM32F407 with M4 core and 42 stepper motor. In addition, we wanted to use unity to control arm through ROS system at first, but we finally found a online source which allowed us to control arm derectly by unity hub, which reduces the complexity and optimizes the performance of the system.

2.2 Control Module

2.2.1 Design Procedure

CAN Module: Through referring to the motor driver board X42_V1.3 instruction manual, it allows us to use the integrated CAN command to control the stepper [4]. However, the STM32F407ZG only has RX and TX pins; the transmission of the CAN signal should be CAN_H and CAN_L two logical voltage [5], which means I need to design a CAN receive and send message module. After searching online, I decided to use the TJA1050 chip to convert the digital CAN signals from the microcontroller to the differential voltage signals required by the CAN bus and vice versa, as shown in the figure 3. The TJA1050 is a CAN transceiver IC that acts as a bridge between the CAN controller (STM32) and the physical CAN bus. It converts single-ended digital signals from the microcontroller into differential signals suitable for transmission over the CAN network, ensuring noise immunity and reliable data communication. At the receiving end, it converts the differential bus signals back into digital signals that can be interpreted by the microcontroller. And to ensure that the bus signals are stable, two 120 **Ω** resistors should be connected in parallel at both ends of the CAN bus [5].

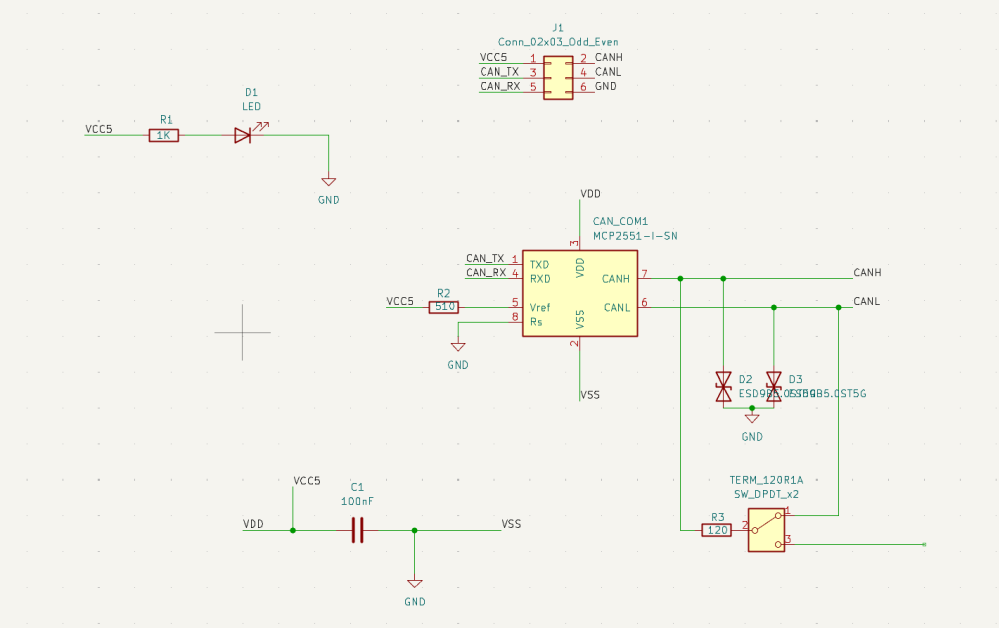

Figure 3: CAN Module

Pressure Sensor:

The FSR402 pressure sensor is a force-sensitive resistor that changes its resistance based on the amount of force applied to it. It is commonly used in applications where measuring pressure or force is required. The more force applied, the lower the resistance. By measuring the voltage across the sensor, we can determine the amount of force being applied. I developed two approaches to read the sensor data:

1. Using the default ADC (Analog-to-Digital) port of STM32F407ZG, which is a 12-bit ADC. The ADC converts the analog voltage signal from the FSR402 into a digital value that can be processed by the microcontroller. The ADC can be configured to read the voltage across the sensor and convert it into a corresponding digital value.

2. Using the LM393 chip to compare the voltage on the pressure sensor with the reference voltage (Variable Resistance). When the voltage on FSR402 is larger than the reference, the output will be high voltage. When the pressure is larger than desired, it will change voltage, which achieves a simple Analog to Digital transform.

For convenience, I designed a circuit integrated on a small PCB 12 and soldered it. The circuit is shown in Figure 4. In the circuit, I connect the FSR402 with a 10kΩ resistor. AO port will output the voltage on FSR402 under 3.3V input, using the $1st$ method. DO port with output the comparison consequence of FSR voltage and 10k Ω variable resistance voltage, using as GPIO input for 2nd method. Finally, I decided to use the 2^{nd} method, and the reason will be explained in the Verification and Test part.

Coding Explain:

Coding in the control module is a huge and most complex part, I will introduce it with two aspects: STM32 and Unity.

In STM32, I develop the code based on the HAL library, which is configured by STM32CubeMX, and implement it in KDM5 (Keil Development Environment).

- can.c: The HAL (Hardware Abstraction Layer) library provides a high-level interface for configuring and using the CAN peripheral. What I did was initialize the handler and change the can_send() function to send a standard CAN message to tell the 42 stepper to run its position circle, like move to which position, how fast, when to stop, and so on. In addition, I also need to write the receive function to receive the position feedback of the motor to control its stop time. For example, if I send 01 36 6B to request the position of the motor, it will respond 01 36 01 00 01 00 00 6B, which

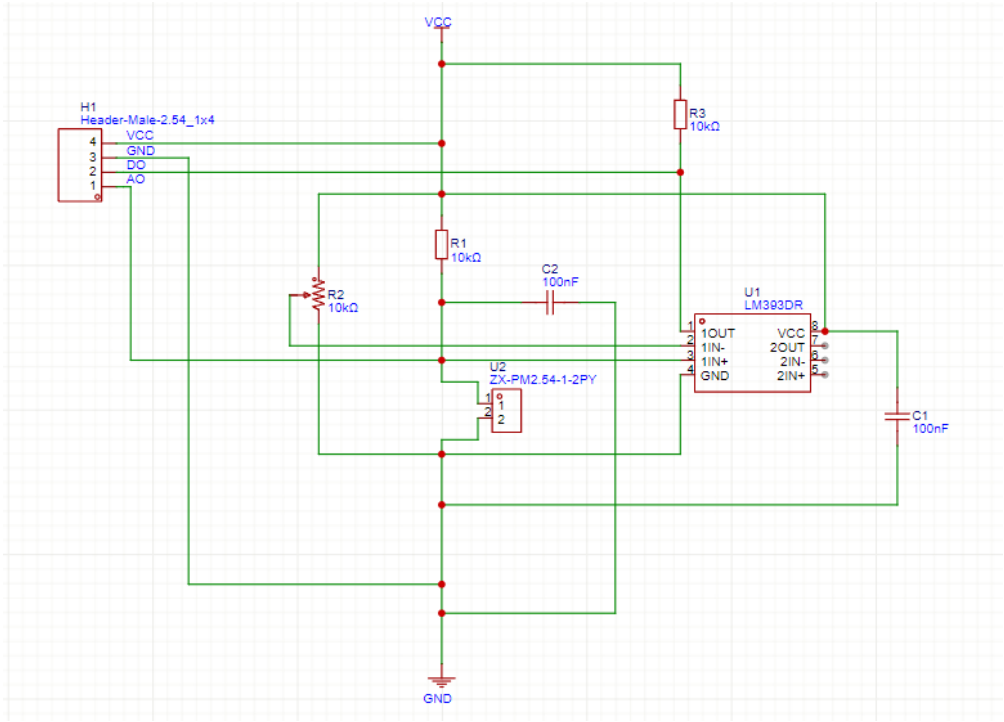

Figure 4: FSR402 Circuit

means the position is 0x00010000 (16), direction is positive. The send function runs smoothly, but the feedback function costs me plenty of time to debug.

- usart.c: The UART (Universal Asynchronous Receiver-Transmitter) peripheral is used for serial communication with the Unity application. The HAL library provides functions to initialize the UART peripheral and rewrite the HAL_UART_RxCpltCallback() function to achieve communication between the STM32 and Unity.

- sensor.c: As I mentioned before, I chose to use the DO as the output of the pressure sensor. To achieve this, I just initialized the GPIN_IN and opened the clock of the corresponding pin, RCC, to receive the digital signal from DO.

- main.c: The main function initializes the system clock, the CAN port, uart port, the sensor, and so on. After initialization, I develop the logic of gripping and releasing in an infinite loop. In the loop, it continuously checks for incoming data from Unity via UART. Unity will send ”S” for start, ”R” for release, and ”H” for stop. After receiving the start, the CAN message will be sent to tell the motor to begin until the reflected pressure sensor data shows the claw has gripped the item. When pressure is over the threshold (input low voltage change), STM32 will send a CAN message to stop the

motor. While Unity sends the release signal, the motor will return according to the position feedback monitoring. Then, the STM32 will wait for the next command from Unity. The following Moore FSM(Figure 5) shows the overall logic.

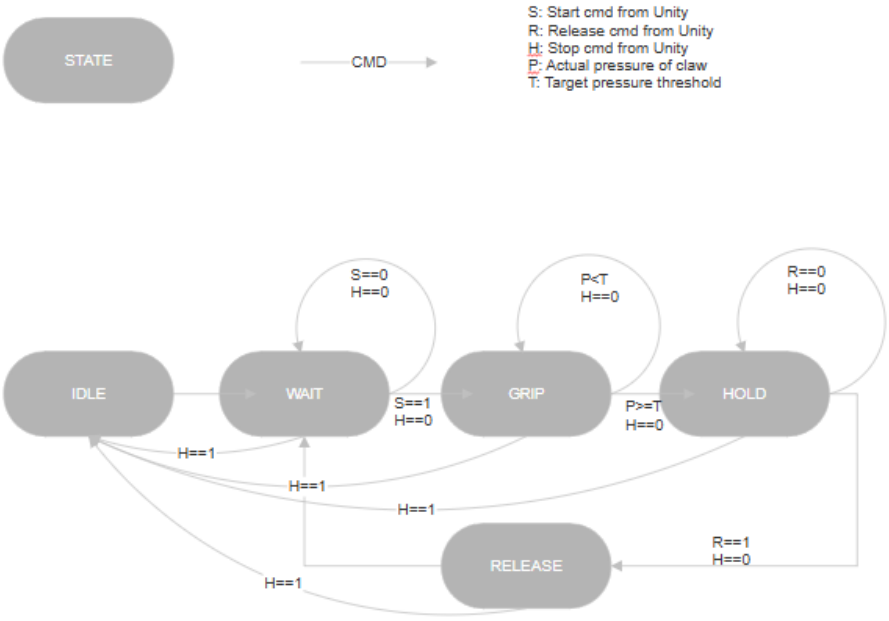

Figure 5: Motor FSM

Unity:

- VR Handler:
- Motor Control:
- Serial Port: The serial port communication in Unity is implemented using the System.IO.Ports namespace. This allows Unity to open a serial port, send data to the STM32, and receive data back. The serial port settings, such as baud rate and parity, are configured to match those of the STM32. For this project, I use the UART line to connect them, and my baud rate was set in STM to 115200, therefore, the serial port in Unity settings corresponds to the baud rate and uses COM4 CH340 PC port.

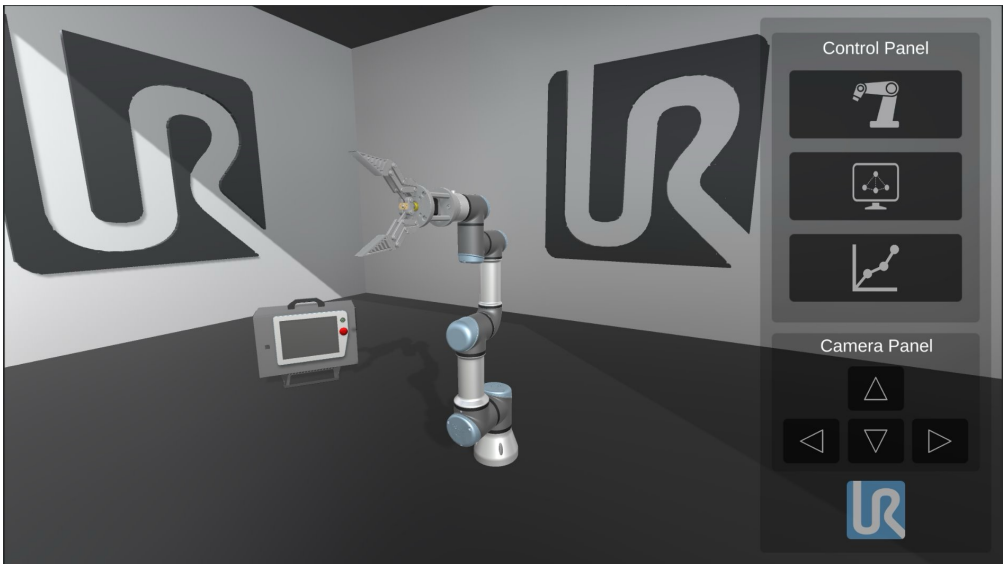

Figure 6: Digital twin of the robot arm and gripper in the Unity simulation environment.

2.3 Unity Digital Twin

2.3.1 Overall Description

The Unity3D Digital Twin Subsystem provides a real-time 3D virtual replica of the physical system, using the Unity game engine to simulate and visualize the system' s state. Its primary purpose is to mirror the physical equipment' s behavior in a virtual environment for monitoring and operator interaction. The subsystem integrates Unity' s physics engine and high-fidelity graphics rendering to model motion. This digital twin receives live data from the physical system via TCP/IP and updates the virtual model accordingly. It maintains synchronization with the real equipment at a frame rate of around 250 frames per second. Within the overall system architecture, the Unity3D Digital Twin acts as the visualization and simulation layer. The Unity3D subsystem consumes these updates and applies them to the virtual model. The digital twin can also produce outputs to control hardware in the real world. The digital twin visualization can run on operator VR headsets.

2.3.2 Implementation Details

The Unity–UR3e robot arm communication is accomplished through TCP/IP, following the official preset communication ports [6]. We set up the Unity connection such that Unity reads data from the robot arm through port 30013, which is a read-only port with a refresh rate of 1000 Hz. Unity writes data to the robot arm through port 30003, which is a read-write

port with a refresh rate of 125 Hz. We use a package for communication that includes two main classes: Stream and Control. The Stream class handles reading data from the robot. The data includes joint orientations in radians and Cartesian positions and orientations in meters. The class uses a thread to continuously read data packets, process them to extract relevant information, and update global variables with the robot's current state to update the Unity model. The Control class manages sending control commands to the robot. It sends commands based on joystick input as byte arrays, which control the robot's speed and movement in Cartesian space. To enable the robot to communicate with an XMLRPC server hosted on a Windows platform, the robot uses a client connection to call remote functions. However, Windows by default blocks such incoming HTTP requests. Therefore, we manually configure the Windows Firewall to allow incoming TCP traffic on ports 30000–30030 to enable communication between the robot and the host server.

For the model construction, learning from previous work [7], we realized a high-fidelity model of the UR3e robot arm, as shown in Figure 6. To provide precise object localization for grasping, we set up two video cameras and streamed them into the scene through the local wireless network. In practice, we used two mobile phones and the iVCam software [8] to stream video to the computer.

As shown in Figure 6, the user interface includes the control panel and the camera panel. The control panel has three functions: the control of the robot arm's X, Y, Z and RX, RY, RZ axes, which allows control of all degrees of freedom of the robot arm; a grasp function linked to the PCB board through a serial port as introduced in Section 2.2; and a linkage function that allows the user to connect to the local robot arm via its IP address. The third part of the panel provides a monitor showing the current state of the robot arm for more precise control. The camera panel provides different camera angles to help the user identify the robot arm' s state.

2.3.3 Design Alternative

An alternative to using two physical cameras was to digitally replicate all graspable objects within the Unity scene and rely solely on the digital twin for object localization. However, we opted for a dual-camera setup streamed via iVCam due to its practicality and cost-effectiveness. This approach offers clearer real-time visuals of diverse physical objects without the need for 3D modeling or object tracking integration. It provides a more intuitive and direct understanding of object position and deformation during grasping, while significantly

reducing development overhead and hardware requirements.

2.4 VR Module

2.4.1 Design Procedure

We developed a VR module to enable real-time interaction with our Unity project on PC. After evaluating multiple linking methods—Meta Quest Link (wired), Air Link (wireless), and Steam Link (cloud wireless)—we selected Steam Link for its balance between low latency and portability. It also eliminates the need for VPN, making it more versatile. For optimal wireless performance, we set up a dedicated router instead of using a phone hotspot.
To enhance user experience and system stability, we upgraded from Oculus Quest 1 (deprecated and unstable) to Meta Quest 3S, enabling better developer support and interaction quality. In the final setup, the VR display is split: the top half shows a Unity control panel (14-button interface for arm control), while the bottom displays live views from two phone cameras for manual calibration. Users control the robotic arm by clicking virtual buttons and referencing the camera feeds for precise adjustments.

2.4.2 Design Details

We designed the VR Module to allow instant connection to our local unity project on the PC. In order to establish stable connection, we have experimented and compared over various linking methods, i.e. cable link, represented by Meta Quest Link; wireless Link, represented by Meta Quest Air Link; and wireless Cloud link, represented by Steam Link. Among all the different methods, cable link ensures the most stable connection and lowest latency. In terms of vision quality and data drop rate, all the methods are pretty comparable. However, what really makes wireless link stand out is that it does not need to be attached to the device. We finally chose Steam Link due to a tradeoff between latency and portability. Besides, the cloud wireless linking method does not require a VPN, so it can be expanded to more application scenarios. As we turn on Steam Link with configurations correctly set, we can see in the VR a real-time screen cast of our PC. More excitingly, we can use both the left and right touch controllers to simulate a cursor that can clicks on the buttons in the unity project, and thus control Robotic Arm through this interface.

Table 1: Comparison of VR-PC Linking Methods

Feature	Quest Link (USB-C)	Quest Air Link (Wi-Fi)	Steam Link (Cloud)
Connection Type	Wired (USB 3.0/3.1)	Wireless (Wi-Fi 5/6/6E)	Cloud (Over Internet)
Latency	Very low, near-native	Low, depends on Wi-Fi quality	High, affected by cloud latency
Visual Quality	Excellent, minimal compression	High, with compression artifacts	Variable, depends on bandwidth
Freedom of Movement	Limited by cable	Full, cable-free	Full, cable-free
Stability	Very stable	Dependent on local interference	Dependent on ISP and network load

For wireless connection, the VR device and our local PC should be connected to the same Wifi. Upon that, in order to further enable smooth connection between VR and unity project, we chose to deploy a router in the laboratory to increase network bandwidth, as opposed to our original choice of phone hotspot.

Furthermore, we even purchased a new VR device for better performance. This is a decision of various factors. First, we initially borrowed Oculus Quest1 from the university, which is outdated and does not support stable cable data transmission. Second, Meta officially shut down its support for Quest1 Developer's mode starting from May 2nd, 2025 [9]. This means it would be difficult to build and test on my own device since then. Therefore, we spent around 2700 yuan to purchase Meta Quest 3S VR online, which later proved to offer much better interaction and development experience.

And as we have mentioned in previous session, we used two separate phone cameras to assist manual calibration of the target position. So our final vision in the VR looks like this: We split the screen into two parts, top and bottom. The top half is a unity project with an interface of 14 buttons (Grip, Release, X+,X-...rotationX+,rotationX-...) for us to directly control the Robotic Arm. The bottom half is further split into two windows of IVCam instances where we can see the phone cameras. When we use the VR touch controller to click onto the buttons, we simultaneously referred to the two cameras for precise position

refinement. And when we want to figure out when is proper moment for lift up, we press 'Grip' button step by step and observe the deformation level of the object. This allows very user-friendly experience with high operation error tolerance.

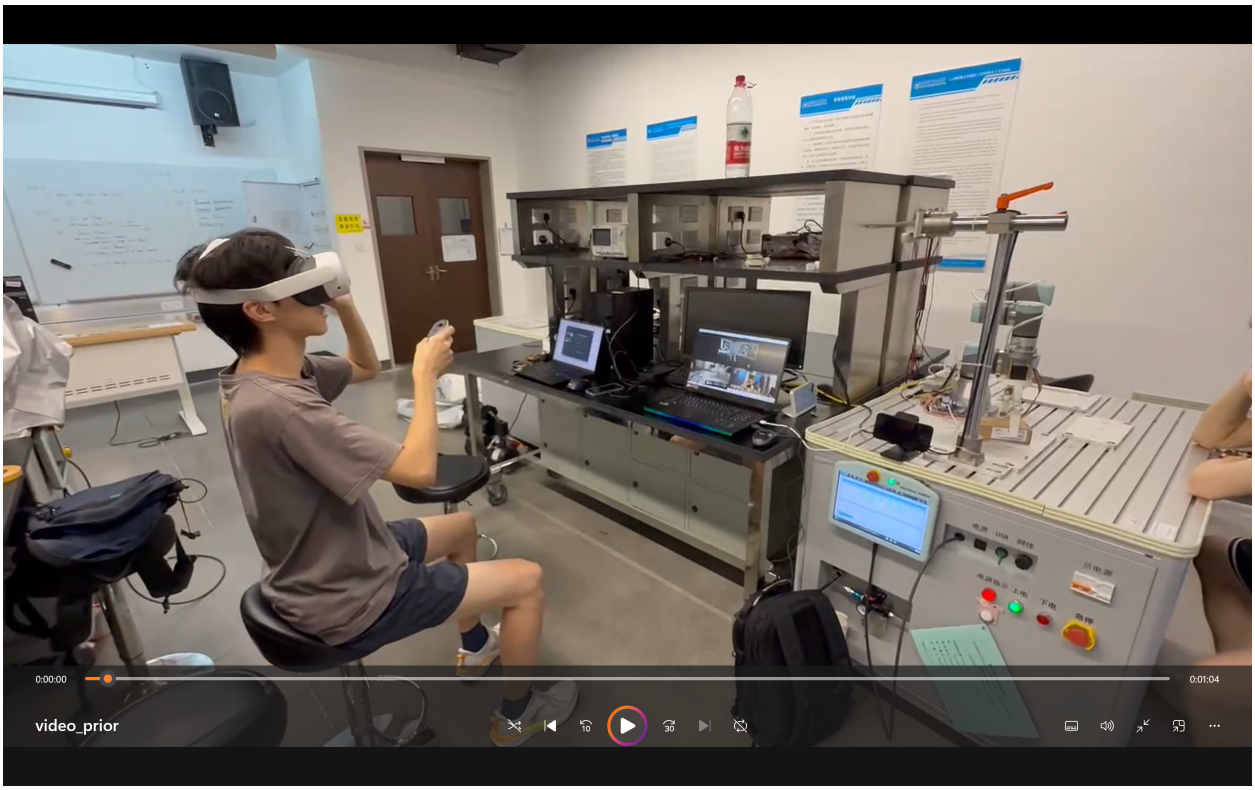

Figure 7: Practice demonstration of VR-controlled robotic grasping system in action.

2.4.3 Verification and Testing

To verify our submodule requirement, ensuring that the VR-Robotic Arm interaction is nearly simultaneous and functions at a practical distance (latency below 1 second; distance beyond sight), we conducted a series of evaluations.

Latency Test: We performed latency tests using both theoretical and practical measurements. The official documentation states a best-case latency of 16.67 ms [10], while our real-world measurement using the UU Accelerator showed an average of approximately 33 ms. This satisfies the requirement for sub-second latency and supports responsive control.

Distance Test: Connectivity tests confirmed stable performance at distances over 20 meters, as limited by the range of our lab-deployed router. This verifies functionality in extended

real-world scenarios, including bedside or remote operations.

Together, these results validate the submodule' s capacity to deliver smooth and responsive VR-based robotic control in diverse settings.

2.5 Gripper Module

2.5.1 Design Procedure

The mechanical claw subsystem serves as the end effector of the UR3 robotic arm, designed to enhance its ability to grasp small objects of irregular shape. We tried to first use the SG90 steering motor as input and finish the prototype shown in Figure 14. By switching to a different thickness of acrylic plate, we were able to solve the problem of the center plate being too thick and causing the servo's POWER output shaft to be positioned too low, which in turn caused the small wheels to rub against the mounting screws.However, due to the inability to obtain precise angular information from the servo horn, coupled with the limited torque of the servo motor and the lack of fine-tuned angle control, we eventually adopted a new mechanical gripper design.

In the latter version, we chose the 42 stepping motor as the power source and took the gripper design in [11]for reference. Compared with the last design, this gripper will be heavier, larger, and more precise. The reference provided a basic structure for us, and we modified and assembled based on that. Compared to the present gripper design, the biggest change is the jaw. We tried a kind of flexible printing material, which is named TPU98A. In the next part, it will be introduced in detail.

2.5.2 Design Details

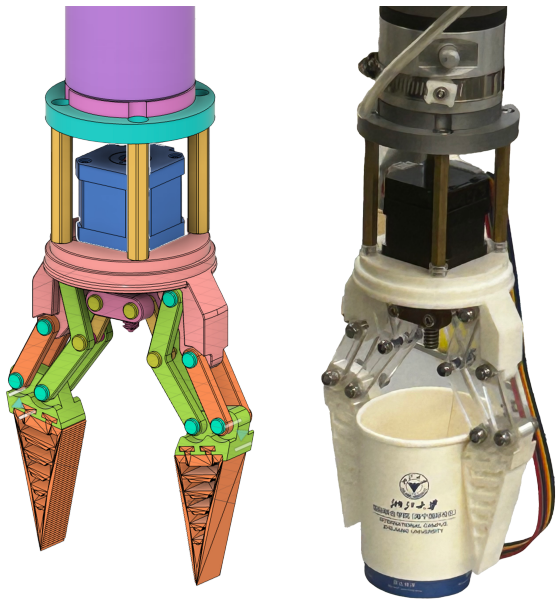

Figure 8: Comparsion between CAD model and final item

As you can see in Figure 8,the gripper subsystem could be divided into several parts.From up to down, they are connecting flange, motor cage, base, actuator flange and TPU jaws.

Component	Description
Connecting Flange	Attaches the gripper to the end of the robotic arm.
Motor Cage	The area where the motor is installed.
Base	Secures the motor and linkage mechanism.
actuator flange	The brown small part. Transmits actuation from the motor to the jaws.
Flexible Jaws	Made by TPU.End-effectors responsible for grasping.

The actuator flange is assembled using M3 heat-set inserts and can slide along the lead screw. It is connected to an acrylic linkage with a thickness of 9.5 mm. The design of TPU flexible jaws is based on [12]. This work proposes a new method of endogenous self-awareness of the flexible gripper based on thin-film bending sensors, which realizes the real-time detection of contact state and object properties by measuring local structural deformation on the load transfer path, and provides a new solution for the environment sensing and interactive operation of adaptive flexible bionic grippers. As for our gripper jaw, we take the fin-ray look for reference and find it works well. The CAD model of the triangular fin-ray jaw is available from [13].

3 Cost & Schedule

3.1 Cost Analysis

Description	Quantity	Price
STM32F407ZGT6 development board	1	¥174.08
Emm42_V5.0 motor control board	1	¥54.99
42mm Stepper motor	1	¥22.99
220V to 24V power supply	1	¥19.80
FSR 402 Pressure Sensor	2	¥20.00
PCB Circuit Board	10	¥43.20
3362P-1-501	5	¥0.48
LM393DR	10	¥0.47
PZ254V-11-04P	20	¥0.29
ZX-PM2.54-1-2PY	5	¥0.27
CGA0603X7R104K500JT	100	¥0.10
0603WAF1001T5E	100	¥0.07
0603WAF1002T5E	100	¥0.07
3362P-1-103	5	¥4.55
Component Courier Fee	2	¥22.00
Total		¥363.37

3.2 Schedule

Table 2: 12-Week Project Schedule by Team Member and Module

Week	Ziming Yan	Jingxing Hu	Jiayu Zhou	Yuchen Yang
1	Research Unity-STM32 communication; study CAN, UART, and FSR sensor usage	Research servo motors, stepper drivers, and claw design materials	Research Meta Quest linking and hand tracking methods	Study digital twin background
2	Finalize control architecture	Sketch mechanical design; select stepper motor and driver	Outline Unity VR scene structure; test Steam Link	Plan Unity scene and 3D model fidelity goals
3	Set up STM32; configure UART/CAN	CAD modeling	Connect Meta Quest via Steam Link	Set up Unity project; import UR3e model
4	Implement UART-Unity interface	Print and assemble first claw prototype	Build Unity VR UI panel	Simulate robot motion in Unity
5	Implement CAN comm layer	Test motor control and gripping logic	Stream camera to Unity via iVCam	Integrate object localization with cameras
6	Integrate FSR402 sensor	Refine grip logic; calibrate servo	Complete VR control input and layout	Sync Unity scene with real-time feedback
7	Build FSM logic in STM32 for grip/release	Improve claw strength and durability	Complete VR panel interaction pipeline	Enable feedback from robot to Unity
8	Debug CAN, UART, ADC timing issues	Tune grip parameters; test stability	Conduct VR usability testing	Reach 250 FPS in Unity
9	Complete control loop	Refine PID + feedback stop logic	Evaluate gesture usability and latency	Integrate Unity twin with physical robot
10	Prepare diagrams and communication logs	Finalize claw model and casing	Finalize VR workflow and UI	Debug full pipeline latency and sync
11	Record STM32 logs; finalize CAN setup	Run grip stress and response tests	Record VR usage demo	Document integration metrics and visuals
12	Live demo of Unity-STM32 closed-loop system	Demo claw grasping various objects	Live VR demo with camera feedback	Present fully synced digital twin system

4 Requirement & Verification

4.1 High-Level Requirement

1, Enable Meta Quest VR to control UR3e Robotic Arm to grasp Objects with different materials properties (Soft and Rigid)
2, Ensure the VR-Robotic Arm Interaction to be nearly simultaneous and distant (Latency below 1 second. Distance: Beyond Sight)
3, Establish Digital Twins of UR3e Robotic Arms in the Unity Project. Enable real-world robotic arm movement with instant in-project simulation (Smooth real-time rendering at 30 FPS)
4, Build 3D-Printed Gripper with Contact Pressure Detector Self-Adaptation, controlled by PCB and driven by Motor (Operate continuously for at least 30 minutes and 6 times of gripping without failure)

4.2 Requirements & Verifications by Subsections

4.2.1 Gripper

This gripper needs to act precisely. However, there are some defects in the present design.As shown in Figure9, the connecting rod that connects to the actuator flange will be subjected to a relatively large tensile force when the claws are closing,they may even broken. The part corresponds to the yellow rods where under the base in Figure8 and they are paired up in pairs to dispersive stress. However, they will break when the gripper is closed tightly, as shown in Figure10.

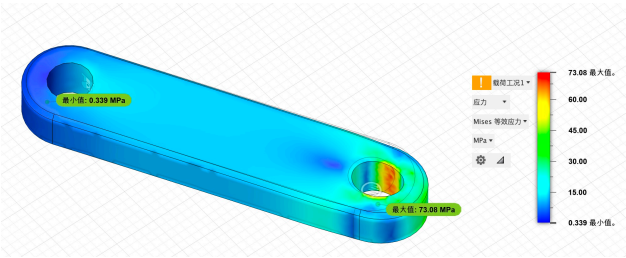

Figure 9: The simulation result of a weak part

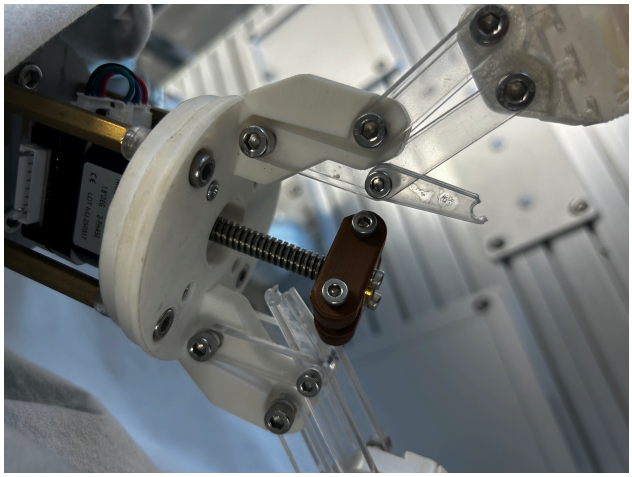

Figure 10: The arcylic rod broke

4.2.2 Control Module

PCB Testing:Before the circuit above, I have designed a failed circuit, which is shown as Figure 13. However, I found errors after the soldering. While applying 3.3V on the VCC port, the voltage on AO didn't change with the pressure on FSR402. Therefore, I used the multimeter to check the circuit. There are errors, like the voltage on AO before FSR insertion should be 0.3V, but I measured 3.17V. In addition, resistance between VCC and DO should be 909.1Ω, but is actually wired at $9.792k\Omega$. Finally, I found there is an open circuit in the PCB design. After designing the new circuit, I succeeded. The following Table 3 shows my measurement history.

Baud Rate Mismatch: While sending the CAN message (position circuit control) to the stepper motor, the motor cannot run correctly as desired. After checking the CAN-related codes, I didn't find the bug. Then, I use an oscilloscope to check the CAN_H port. Normally, if the CAN message is sent correctly, the CAN_H will have a voltage change; the low voltage should be about 0, and the high should be 5V. However, I found that the CAN_H wave is always 4.176 mV, which means the CAN initialization failed. So I check the default CAN HAL init, the baud rate is set to 1M, which is mismatched from my motor's CAN receive rate. My designed baud rate should be 50000. The CAN bus baud rate on STM32 is determined by configuring the CAN timing registers, primarily the Prescaler, BS1, BS2, and SJW parameters, and the rate computation should use the equation:

$$\text{Baud Rate} = \frac{\text{CAN Clock Frequency}}{\text{Prescaler} \times (1 + \text{BS1} + \text{BS2})} \tag{1}$$

For my project, system clock is 168Mhz, the CLK OF CAN is APB1 CLK $= \frac{\text{System Clock}}{4} = 42Mhz$ I changed the Prescaler to 14, BS1 to 4, BS2 to 1. Therefore my designed baud rate should be $\frac{42Mhz}{14\times(1+4+1)} = 50000$, which matched the default rate. After changing the baud rate, I successfully finished the communication with the motor.

STM32 ADC Delay:After comparing my two methods prepared to monitor the pressure by FSR, I found that the DO port responds to the pressure threshold faster compared with AO after ADC. I thought the DO would enable my claw to be more sensitive, but actually, the ADC delay is about 1ms, which is not acceptable for my project. Therefore, I choose to use the DO port as input.

CAN Feedback:At first, I didn't think the position feedback of the motor was necessary. However, I found that there are errors in the phase of motor D axle, varying from 0.9° to 1.2°. With accumulation, the original position of the claw will be changed, causing the claw cannot close completely. Therefore, I added the CAN feedback function to monitor the position of the axle, ensuring the motor runs backward until the position is 0. Through testing, the average position error decreased, within 0.08°.

4.2.3 Digital Twin

Real-Time Visualization Performance: The subsystem is required to provide a smooth real-time visualization of the physical system. It must maintain a frame rate of at least 30 frames per second (FPS) during normal operation, with a target of 60 FPS for optimal smoothness. In practice, we achieved an average of over 250 FPS, as shown in Figure 11. Data Throughput and Update Rate: The Unity3D subsystem shall handle incoming data streams from the physical system at a sufficient rate. It is expected to support at least a 10 Hz update frequency for all critical sensor inputs. In practice, we achieved an update frequency of 1000 Hz for input data in the Stream thread from the robot arm, and an update rate of 125 Hz for output control data. Scene Complexity and Rendering Capability: The subsystem shall support the full complexity of the system' s 3D model. It should render all relevant components with high visual fidelity. As shown in Figure 6, we achieved high-fidelity models for both the robot arm and the claw. Integration and Compatibility: The Unity3D subsystem must integrate seamlessly with the overall system' s data and control architecture. We integrated the whole system as expected; the demo demonstrates our successful integration.

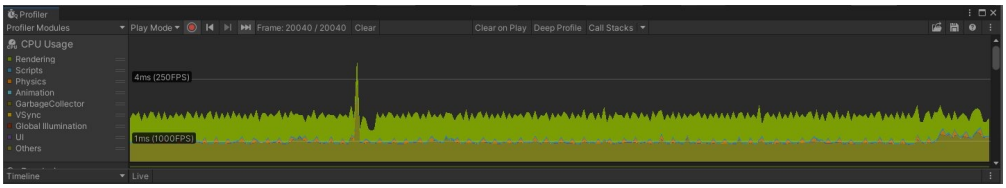

Figure 11: Frame rate performance of the Unity3D Digital Twin subsystem, demonstrating an average of over 250 FPS during operation.

4.2.4 VR

To validate the VR system' s capability to control the UR3e Robotic Arm for grasping objects with different material properties, we conducted a series of real-time manipulation tests. Objects with distinct mechanical characteristics—such as soft sponges and rigid plastic blocks—were placed within reach of the robotic gripper. The VR interface enabled users to precisely control grip force and timing via a 14-button Unity panel. For soft objects, gradual activation of the "Grip" button allowed the user to visually assess deformation through live IVCam video feeds and stop gripping at an appropriate moment to avoid damage. For rigid objects, full grip closure was executed without material failure. These tests demonstrate the system' s robustness across a range of material compliance levels.

We tested the robotic arm via VR to grasp both soft (e.g., sponge) and rigid (e.g., plastic) objects. Users adjusted grip force in real time using the VR interface while observing deformation through live camera feeds, ensuring successful and damage-free manipulation.

Latency was measured at 33 ms using UU Accelerator, well below the 1-second threshold. The system also maintained stable performance beyond 20 meters using a dedicated router, confirming reliable long-distance control.

5 Conclusion

5.1 Summary of Completion

We successfully developed a VR-controlled robotic grasping system that enables intuitive and remote interaction with physical objects. The system allowed users to grasp and release four different objects—paper cup, medicine box, sponge, and bandit—with high precision using Meta Quest VR and Unity integration. Real-time control was achieved with low latency and stable wireless streaming via Steam Link. Users could perform step-by-step gripping actions while observing object deformation through IVCam feedback, enabling safe handling of both soft and rigid items. The project validates the feasibility and potential of immersive human-robot interaction for assistive and remote applications.

5.2 Further Improvement

While our current system demonstrates robust VR-controlled robotic manipulation, several hardware and software enhancements can further expand its functionality and user experience.

Hardware Enhancements:

- Three-Jaw Gripper Design: Upgrading to a three-jaw configuration will improve contact stability and object adaptability, especially for round or asymmetric shapes.
- Larger Sawteeth or Surface Grip Features: Enhancing the claw's surface with larger sawteeth or textured material will increase friction and reduce slippage when handling smooth objects.
- Metal Linkages: Replacing plastic joints with metal linkages will significantly improve structural integrity, precision, and safety under stress.

Software Enhancements:

- Automatic Trajectory Planning: Integrating autonomous motion planning algorithms will reduce the need for manual adjustments and improve efficiency in complex tasks.
- Controller Integration: Mapping directional control buttons directly onto the VR touch controllers will simplify input and enhance real-time responsiveness.

- Standalone Unity App: Building the Unity project as a self-contained, publishable application will support broader deployment and usability across VR platforms.

These future developments aim to enhance the system' s adaptability, reliability, and accessibility in both assistive and industrial contexts.

5.3 IEEE Ethical Standards Compliance

Throughout the development of this project, we have adhered strictly to the IEEE Code of Ethics to ensure academic integrity, technical honesty, and safety in engineering practice. In particular, our mechanical gripper design is based on a publicly shared model by Raunak Jain. We properly credited the original work and documented all modifications made to enhance adaptability and application in assistive environments. These improvements reflect our commitment to building upon prior contributions while maintaining transparency and accountability.

We also prioritized safety in all fabrication and testing procedures. All team members received training before operating high-risk equipment such as laser cutters and robotic arms. We implemented rigorous safety protocols during acrylic sheet processing, bonding, and mechanical assembly to avoid accidents and ensure safe working conditions. Our adherence to ethical and safety standards exemplifies responsible engineering conduct and reinforces our commitment to protecting users, collaborators, and intellectual property throughout the project lifecycle.

5.4 Broader Impacts

The integration of Virtual Reality with robotic manipulation in our system offers significant potential across multiple domains. In healthcare, it enables bedridden patients to interact with their environment independently, reducing reliance on caregivers and improving quality of life. In industrial settings, the system provides a safe and effective solution for remote manipulation, particularly valuable during emergencies or pandemics where human presence is limited or risky. Furthermore, its intuitive VR interface lowers the barrier for non-expert users to operate complex robotic systems, promoting broader accessibility in education, rehabilitation, and teleoperation. As VR and robotics technologies continue to advance, our project lays foundational work for scalable, human-centric robotic assistance in both personal and professional contexts.

References

[1] Meta Developers, Hello vr: Unity tutorial for meta horizon platform, Accessed: 2025-05-25, n.d. [Online]. Available: https://developers.meta.com/horizon/documentation/unity/unity-tutorial-hello-vr/.

[2] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," IEEE Transactions on Industrial Informatics, vol. 15, no. 4, pp. 2405–2415, 2019. doi: 10.1109/TII.2018.2873186.

[3] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in Transdisciplinary Perspectives on Complex Systems: New Findings and Approaches, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds. Cham: Springer International Publishing, 2017, pp. 85–113. doi: 10.1007/978-3-319-38756-7_4. [Online]. Available: https://doi.org/10.1007/978-3-319-38756-7_4.

[4] 张大头, 步进闭环驱动说明书 rev1.3, https://blog.csdn.net/zhangdatou666/article/details/132644047, Accessed: 2025-05-18, 2023.

[5] 艾格北峰, Can 总线协议, https://blog.csdn.net/qq_35057766/article/details/135580884, Accessed: 2025-05-18, 2024.

[6] Universal Robots, Overview of client interfaces, Accessed: 2025-05-25, n.d. [Online]. Available: https://www.universal-robots.com/articles/ur/interface-communication/overview-of-client-interfaces/.

[7] R. Parak, A digital-twins in the field of industrial robotics integrated into the unity3d development platform, https://github.com/rparak/Unity3D_Robotics_Overview, 2020–2024.

[8] e2eSoft, Ivcam - use mobile phone as a pc webcam, Accessed: 2025-05-25, n.d. [Online]. Available: https://www.e2esoft.com/ivcam/.

[9] Meta Developers, Changes coming to quest 1 in 2023 - meta, Accessed: 2025-05-25, 2023. [Online]. Available: https://developers.meta.com/horizon/blog/changes-coming-quest-1-2023-meta/.

[10] Valve Corporation, Steam link on steam, Accessed: 2025-05-25, n.d. [Online]. Available: https://store.steampowered.com/app/353380/Steam_Link/.

[11] Muhmmd Abu Baker, Robot servo motor gripper, Accessed: 2025-05-29, 2023. [Online]. Available: https://grabcad.com/library/robot-servo-motor-gripper-2.

[12] G. Chen, S. Tang, S. Xu, et al., "Intrinsic contact sensing and object perception of an adaptive fin-ray gripper integrating compact deflection sensors," IEEE Transactions on Robotics, vol. 39, no. 6, pp. 4482–4499, 2023. doi: 10.1109/TRO.2023.3311610.

[13] EddieChan, Dummy 末端执行器, Accessed: 2025-05-25, n.d. [Online]. Available: https://makerworld.com.cn/zh/models/249697-dummymo-duan-zhi-xing-qi-___rou-xing-jia-zhua?from=search.

Appendix A Supplementary Materials

A.1 Codes

A.1.1 STM32F407 Motor Control Logics

```c
#include <stdio.h>
#include "MOTOR.h"
#include "delay.h"
#include "PWM.h"
#include "sys.h"
#include "SENSOR.h"
#include "usart.h"
#include "can.h"
#include "Emm_42.h"
#include "ROS.h"
#include <stdbool.h>

bool grip_cmd = false;
bool release_cmd = false;

int main(void)
{
    uint8_t data_receive[12];
    uint8_t len;
    uint32_t id;
    typedef enum {
        CLAW_IDLE = 0,
        CLAW_WAIT,
        CLAW_GRIPPING,
        CLAW_HOLDING,
        CLAW_RELEASING,
        GET,
    } CLAW_STATE;
    CLAW_STATE State = CLAW_IDLE;

    // 系统初始化
    Stm32_Clock_Init(336, 8, 2, 7);
    delay_init(168);
    uart_init(115200);
    ADC_Init();
```

```
    Sensor_Init();
    CAN_Init();
    //USER_CAN1_Filter_Init();
    // 打开中断接收

    Emm_V5_En_Control(0x01, true, false);   // 使能电机
    HAL_Delay(100);

    Emm_V5_Modify_Ctrl_Mode(0x01, false, 2); // 设置为闭环控制模式（模式 2）
    HAL_Delay(100);

    printf("Init Finish\r\n");

    while (1)
    {
        uint8_t reset = 0;
        int32_t pos = 0;
        Check_ROS_Command();

        switch (State)
        {
            case CLAW_IDLE:
                Emm_V5_Reset_CurPos_To_Zero(0x01);  // 初始设为 0
                printf("State: CLAW_IDLE\r\n");
                State = CLAW_WAIT;  // 等待命令
                HAL_Delay(50);
                break;

            case CLAW_WAIT:
                if (grip_cmd)
                {
                    grip_cmd = false;
                    Emm_V5_Pos_Control(0x01, 0, 100, 0, 1000, false, false);
                    printf("State: CLAW_GRIPPING\r\n");
                    State = CLAW_WAIT;
                }
                else if (release_cmd)
                {
                    State = GET;
                }
                HAL_Delay(50);
```

```
77                  break;
78
79              case CLAW_GRIPPING:
80                  if (Pressure_Trigger())
81                  {
82                      Emm_V5_Stop_Now(0x01, false);
83                      ToROS_Message("GRIP_COMPLETE\r\n"); // send complete signal to
                        ↪ ROS
84                      printf("State: CLAW_HOLDING\r\n");
85                      printf("Wait for Release cmd ...\r\n");
86                      State = GET;
87                  }
88                  HAL_Delay(50);
89                  break;
90
91              case GET:
92                  while (1)
93                  {
94                      Emm_V5_Read_Sys_Params(0x01, S_CPOS);
95                      CAN_Receive_Message(&id, data_receive, &len);
96                      if (data_receive[0] == 0x36) // 确保是位置反馈
97                      {
98                          pos = (data_receive[2] << 24) | (data_receive[3] << 16) |
99                                (data_receive[4] << 8) | data_receive[5];
100
101                         if (data_receive[1] == 0x01) pos = -pos;
102                         printf("Current position: %d\r\n", pos);
103                         break;
104                     }
105                     else
106                         printf("loop\r\n");
107                 }
108                 State = CLAW_HOLDING;
109                 break;
110
111             case CLAW_HOLDING:
112                 if (release_cmd)
113                 {
114                     release_cmd = false;
115                     Emm_V5_Pos_Control(0x01, 1, 100, 0, pos, true, false);
116                     printf("State: CLAW_RELEASING\r\n");
```

```
117                     State = CLAW_RELEASING;
118                 }
119                 HAL_Delay(50);
120                 break;
121 
122             case CLAW_RELEASING:
123                 while (reset != 1)
124                 {
125                     Emm_V5_Read_Sys_Params(0x01, S_CPOS);
126                     CAN_Receive_Message(&id, data_receive, &len);
127                     if (data_receive[0] == 0x36)
128                     {
129                         pos = (data_receive[2] << 24) | (data_receive[3] << 16) |
                            ↪  (data_receive[4] << 8) | data_receive[5];
130                         if (data_receive[1] == 0x01) pos = -pos;
131                             printf("Current position: %d\r\n", pos);
132 
133                             if (pos >= -5 && pos <= 5)
134                             {
135                             reset = 1;
136                             Emm_V5_Stop_Now(0x01, false);
137                             printf("Motor has returned to zero.\r\n");
138                             }
139                     }
140                 }
141                 State = CLAW_WAIT;
142                 //              Emm_V5_Reset_CurPos_To_Zero(0x01);
143                 ToROS_Message("RELEASE_COMPLETE");
144                 printf("State: CLAW_RELEASED\r\n");
145                 printf("Wait for next cmd ...\r\n");
146                 HAL_Delay(50);
147                 break;
148 
149                 State = CLAW_WAIT;
150                 ToROS_Message("RELEASE_COMPLETE");
151                 printf("State: CLAW_RELEASED\r\n");
152                 printf("Wait for next cmd ...\r\n");
153                 HAL_Delay(50);
154                 break;
155 
156             default:
```

```
157                 State = CLAW_IDLE;
158                 break;
159         }
160         HAL_Delay(50);
161     }
162 }
163
164 void Error_Handler(void)
165 {
166     printf("Error\r\n");
167 }
168
```

A.2 Figures

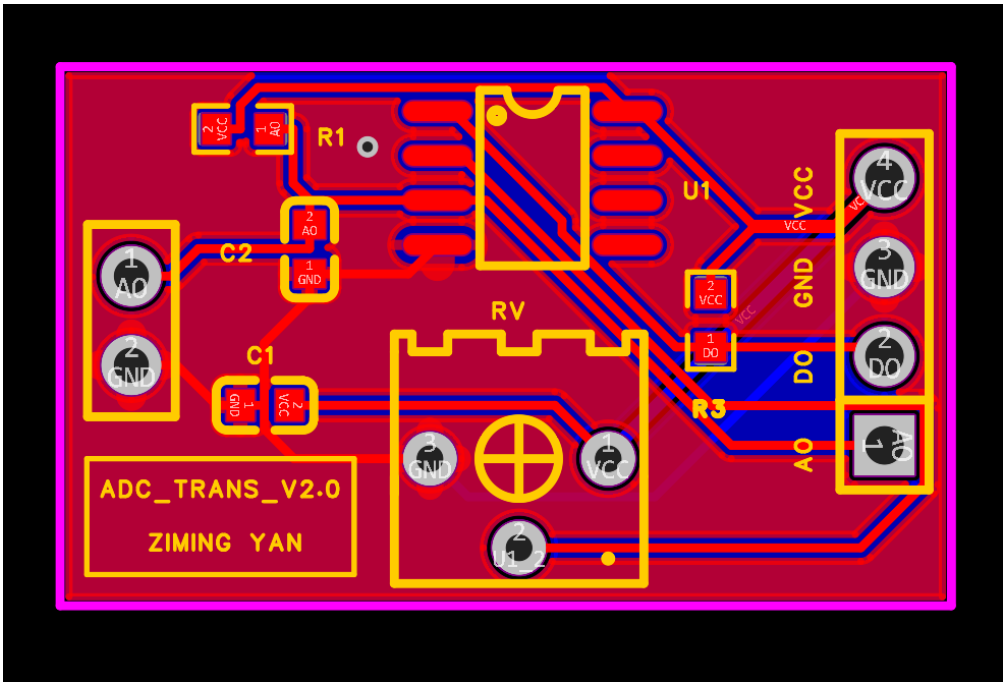

Figure 12: FSR402 PCB

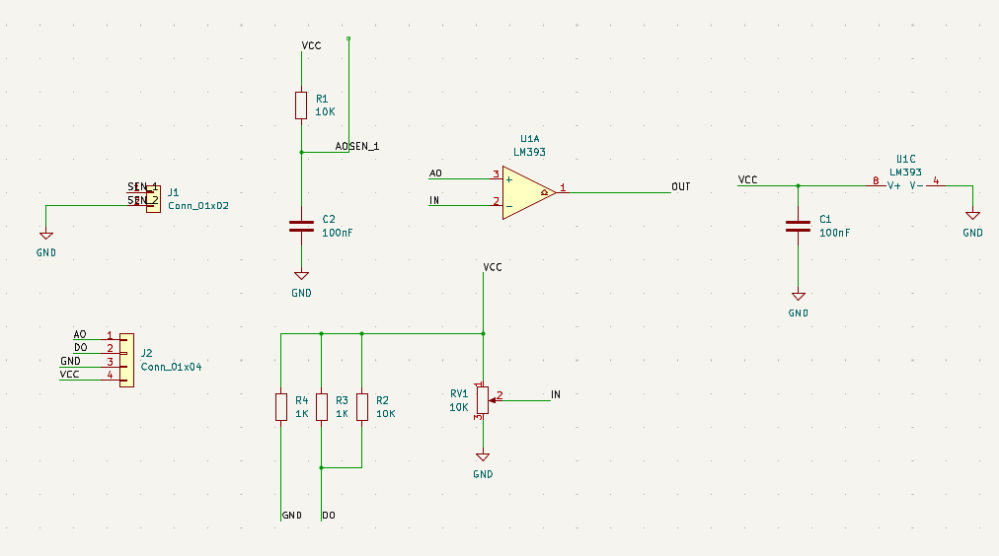

Figure 13: FSR402 Failed Circuit

A.3 Tables

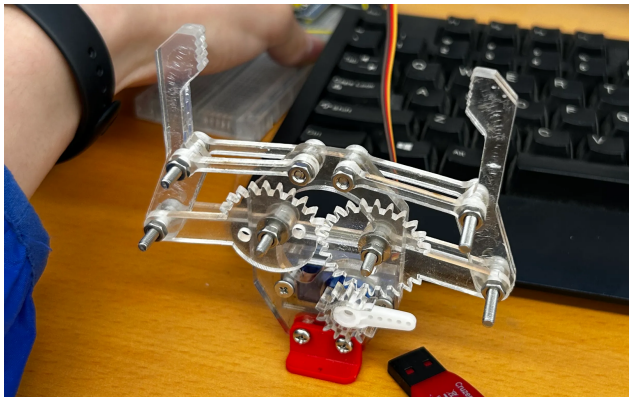

Figure 14: first prototype gripper with Steering motor

PCB Type	Test Item	Expected Result	Actual Result
Failed	Voltage on AO	$\frac{1k\Omega}{11k\Omega} \times 3.3V = 0.3V$	3.17V
Failed	R between sensor port 2 and VCC	$10k\Omega$	$9.792k\Omega$
Failed	R between VCC and DO	$\frac{1k\Omega \times 10k\Omega}{1k\Omega + 10k\Omega} \approx 909.1\Omega$	$6.147k\Omega$
Success	Voltage on AO	3.3V	3.197V
Success	R between sensor port 1 and VCC	$10k\Omega$	$9.807k\Omega$
Success	R between VCC and DO	$10k\Omega$	$9.076k\Omega$

Table 3: PCB Testing