ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Grasping Any Object with Robotic Arms with Language Instructions

---

Team #10

JUNZHOU FANG
(junzhou5@illinois.edu)
JUNSHENG HUANG
(jh103@illinois.edu)
ZIXIN ZHU
(zixinz6@illinois.edu)
ZIXUAN ZHANG
(zixuan21@illinois.edu)

Professor: Gaoang Wang, Liangjing Yang
TA: Tianci Tang, Tielong Cai

May 26, 2025

Contents

1 Introduction

1.1 Problem

Patients who are bedridden or have severe mobility impairments—whether from stroke, spinal cord injury, or neuro-degenerative disease—often struggle with simple tasks such as reaching for water, medication, or personal devices. Their reliance on caregivers reduces autonomy and increases workload in clinical settings. Meanwhile, recent studies show that assistive robots have begun to ease this burden and improve outcomes [1] [2]. However, most current devices cannot parse natural language or operate reliably amid the clutter and tight spaces at a bedside. A next-generation system must combine accurate speech understanding, real-time object recognition, and precise manipulation while generalizing to many everyday objects without lengthy retraining. Such an advance would restore greater independence to patients and lighten the demands placed on caregivers.

1.2 Solution

Our expected solution is a smart robotic arm equipped with a well-designed recognition system based on computer vision (CV) and natural language processing (NLP). The block-diagram of our design is shown in Figure 1. To be more specific, our design accepts three input: a human language instruction from microphone or keyboard, a RGB image and a depth image taken by the depth camera, and gives two outputs: the destination angle of each of the six motors and target label of the object printed on printed circuit board (PCB). Eventually, the destination angles can guide the robotic arm moving to the correct place.

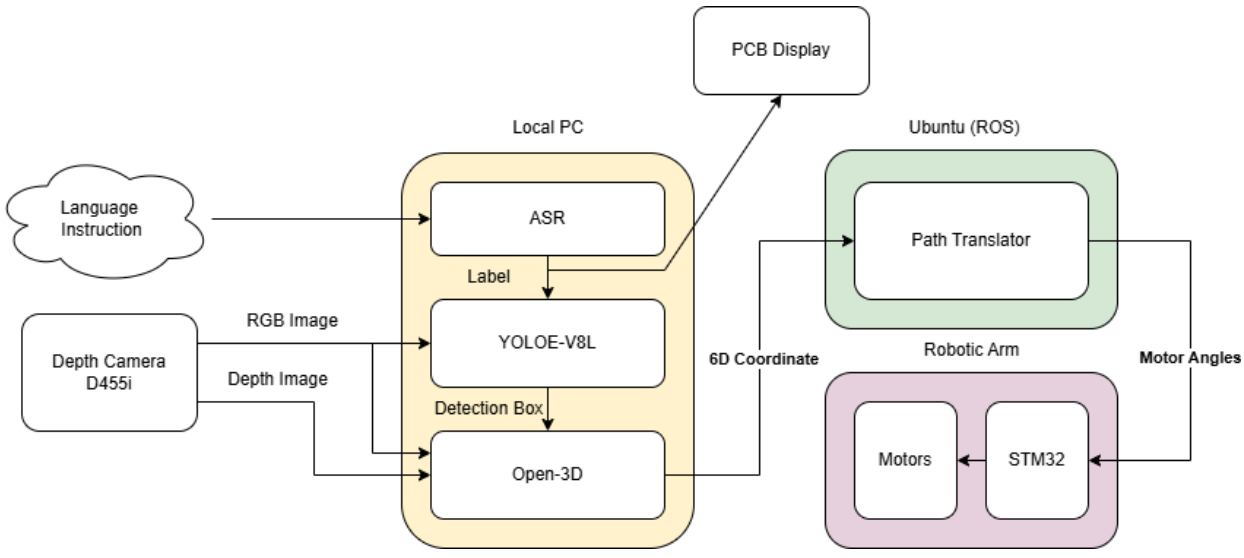


Figure 1: High-level block diagram of our design

Our design is divided into six major modules: automatic speech recognition (ASR), CV,

6d-pose generator, PCB display, path translator, and robotic arm. The performance requirement of each modules are:

- **ASR**: Translate human-language input into a string and identify the name of the target object. We use Wave2Vec 2.0 as the ASR model and add a NLP model after this to extract the label.
- **CV**: Use the RGB image taken from the depth camera, identify the target object using a detection box of pixel-level coordinate. We use YOLOE-V8L.
- **6D-Pose Generator**: Utilize both RGB image and depth image from the depth camera, determine the 6D coordinate, [x,y,z] and three Euler angles for the object in the detection box. We use Open-3D.
- **PCB Display**: Display the name of the target object on a 0.96-inch OLED screen.
- **Path Translator**: Use the 6D coordinate of the target object to choose the optimal grasping position, and map the position to a set of destination angles of the motors.
- **Robotic Arm**: Use YahBoom DOFBOT SE robotic arm with six motors driven by an STM32 controller to grab objects based on the output of the path translator.

The block diagram changes a lot throughout the semester, as we give up running AI models on remote servers or on Ubuntu. The reason is that we cannot establish a web link to the remote server due to the firewall issue, and Ubuntu does not support USB drivers for the depth camera and microphone.

2 Design

2.1 ASR (Wave2Vec 2.0)

The primary function of this automatic speech recognition (ASR) model is to transform user-provided instructions (regardless of input text or speech signals) to exact object labels. These labels serve as triggers for the YOLOE (CV model) to perform designated tasks. This module acts as a critical bridge between language understanding and visual perception in a multi-modal system, enabling instruction-driven control via natural language. Specifically, this model would be divided into two components.

The first component is the Automatic Speech Recognition (ASR) module, which is responsible for processing incoming audio signals. This module would first improve speech quality through noise reduction and signal enhancement. Then, The refined audio is transcribed into its corresponding textual representation, thereby serving as the input for subsequent natural language processing modules. We choose a lightening framework from Meta AI: Wav2Vec 2.0 [3]. It is a self-supervised speech recognition framework that learns powerful speech representations directly from raw audio using Transformer architectures and contrastive learning. Figure 2 illustrates the workflow of the framework.

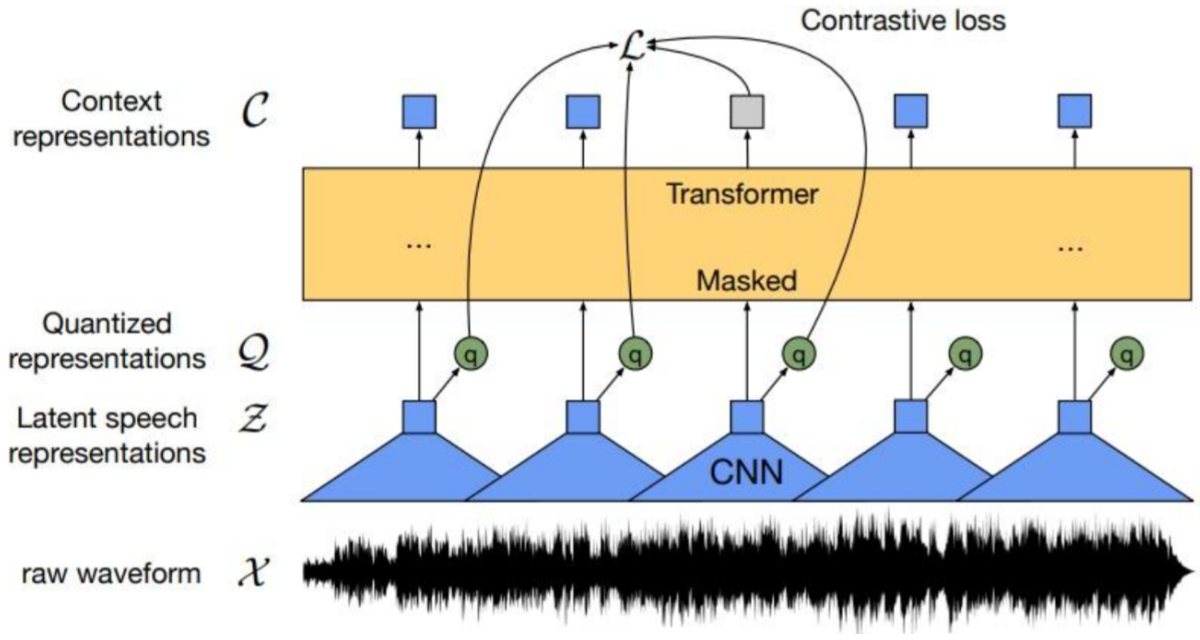


Figure 2: Illustration of Wav2Vec 2.0 framework

The second component is the Semantic Label Extraction module, which processes the textual instructions obtained from the ASR module or directly from the user. It employs predefined regular expression rules to extract key terms that correspond to the target object for grasping, serving as the label for the downstream computer vision module. Recognizing the limitations of rule-based keyword extraction in handling the diversity and ambiguity of natural language, we also consider leveraging semantic similarity models

to infer potential target labels. This hybrid approach enhances the robustness and generalization of instruction interpretation.

2.2 CV (YOLOE-V8L)

We use YOLOE-V8L as our visual detection model. The model is capable for recognizing all objects in the image by extracting them from raw image and comparing them with predefined vocabulary base [4]. Since we already get the instruction from ASR, we produce a text prompt and just ask the model to select the target object accordingly. The output of the model will be a detection box with pixel-level coordinates and confidence.

In our implementation, we integrate the code of controlling the depth camera with the code of running YOLOE inference. As we begin reading the continuous picture in the depth camera, we can press key `C` to store a frame and send the frame to YOLOE inference. To improve the robustness, we ask the model to only return the detection box with highest confidence. A more rigerous pseudocode is:

Algorithm 1 Run_YOLOE_Inference_Once

Ensure: {color-image path, depth-frame array, detected box, depth-scale}
Initialize RealSense pipeline with color and depth streams
while true **do**
    Capture current frames and show live color preview
    if user presses `C` **then**
        Save the current color and depth frames to *work_dir*
        Call YOLOE_INFERENCE to obtain *box* with highest *confidence*
        return collected outputs and exit loop
Stop the camera and close the window

There are several YOLOE models differing in size and performance. We finally choose YOLOE-V8L, the largest YOLOE model with best performance, given that it only takes about 5G on GTX3090 to do a single image reference. A detailed table of model differences and performance comparison can be found in Appendix A.

2.3 6D-Pose Generator (Open-3D)

To enable the clamp on the robotic arm to successfully grasp an object, we need a 6D coordinate that not only describes the location, but also the pose of the object. Typically, a 6d coordinate is an array with length of six: $[X, Y, Z, Roll, Pitch, Yaw]$, where the first three dimensions describe the location and the last three dimensions, usually referred as Euler Angles, tell the rotation. We select Open-3D to generate the 6d coordinate. Open3D is an open-source library designed for fast, easy manipulation of 3-D data [5]. The inputs of this library are RGB and depth image, detection box, and some intrinsic calibration parameters in depth camera.

In our project, we use an Intel RealSense D455i depth camera mounted in a fixed position. For this depth camera, we do not need to do intrinsic calibration by ourselves. Instead, we can directly access these intrinstic parameters using a light-weight python scripts. Table1 records the intrinsic calibration parameters for D455i.

Table 1: RealSense D455i intrinsic calibration (640×480)

Parameter	**Value**
Image width	640
Image height	480
f_x	390.57965
f_y	390.57965
c_x	320.92871
c_y	240.26723
Distortion model	`brown_conrady`
Distortion coefficients	$\begin{bmatrix}0, & 0, & 0, & 0, & 0\end{bmatrix}$
Camera matrix K	$\begin{bmatrix}390.57965 & 0 & 320.92871 \\ 0 & 390.57965 & 240.26723 \\ 0 & 0 & 1\end{bmatrix}$

We implement our 6D-pose generator based on Open-3D on our own, and the code workflow is summarized in pseudocode 2.

Algorithm 2 SavePoseFromArrays

Require: RGB file path *rgb_path*, depth array *depth_path*,ROI box (x_0, y_0, x_1, y_1), depth scale s,intrinsics $\{f_x, f_y, c_x, c_y\}$,output file *pose_path*

rgb ← CV2.IMREAD(*rgb_path*)
depth ← *depth_path* ▷ 16-bit metres
Build binary *mask* with ones inside ROI
Convert RGB + depth to OPEN3D `RGBDImage`
Create pinhole intrinsics object using f_x, f_y, c_x, c_y
Generate full point cloud P from RGB-D image
$P_{\text{roi}} \leftarrow$ points in P where *mask*$= 1$
Remove zeros; voxel-down-sample P_{roi}
Obtain oriented bounding box (OBB) of P_{roi}
$\mathbf{t} \leftarrow$ OBB centre (mm);$\mathbf{R} \leftarrow$ OBB rotation matrix
$\boldsymbol{\phi} \leftarrow$ EulerXYZ($\mathbf{R}$) ▷ roll, pitch, yaw (rad)
Compose JSON object with **position** $\mathbf{t}$ and **orientation** $\boldsymbol{\phi}$
Write JSON to *pose_path*
return $\mathbf{t}$, $\boldsymbol{\phi}$

2.4 PCB Display

To provide a more intuitive and user-friendly interface for object identification during grasping operations, we designed a PCB display system. This system features a 0.96-inch ZJY096I0400BG01 OLED screen driven by a STM32F103C6T6 microcontroller, and communicates with the PC-side ROS module via USB. This system can display the name of the object currently being grasped, providing immediate visual feedback to the user. Figure 3 illustrates the circuit diagram of the PCB display system.

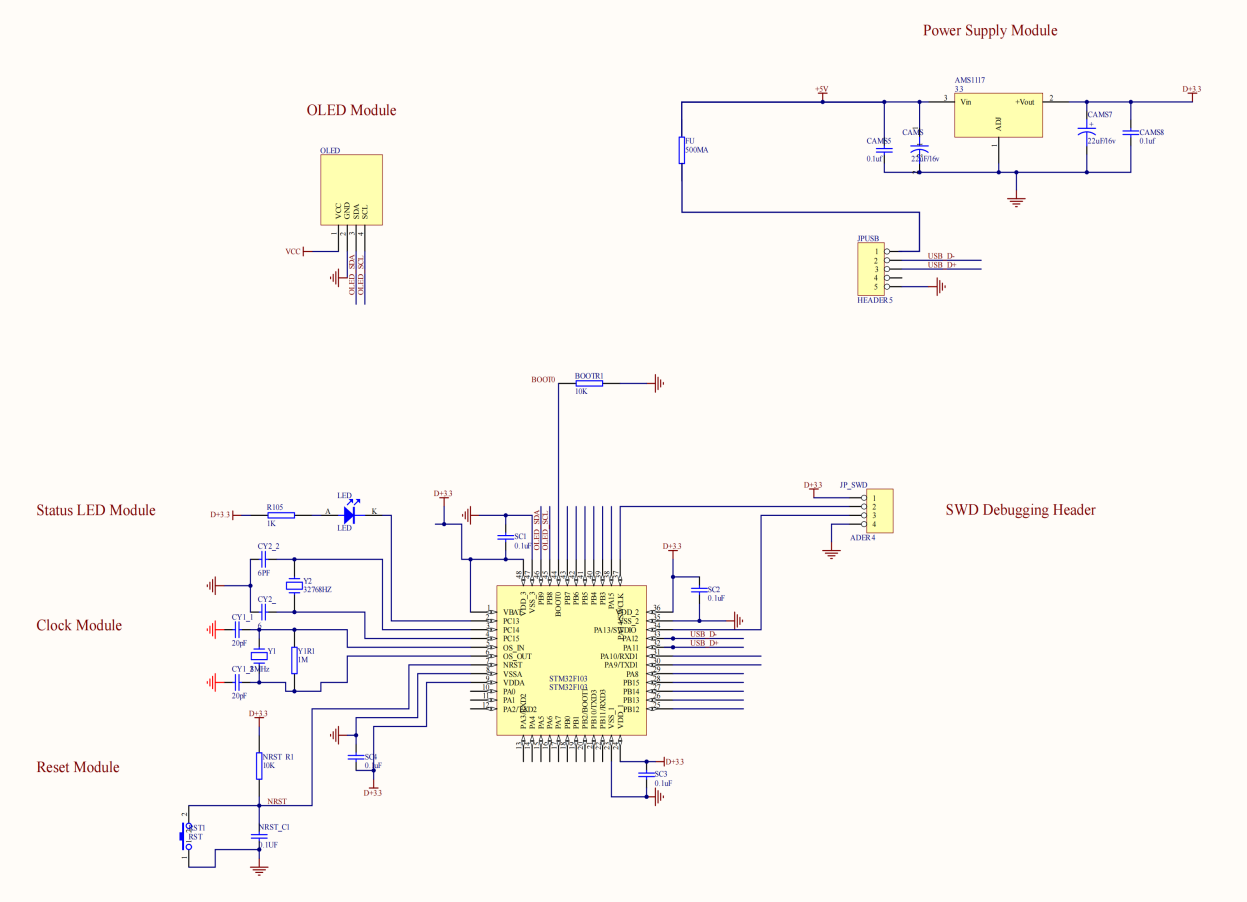


Figure 3: Circuit diagram of the PCB display system

The system is composed of several functional modules as described below.

- **STM32F103C6T6 Microcontroller**
  This microcontroller serves as the central control unit of the system. Based on an ARM Cortex-M3 core running at 72 MHz, it integrates 64 KB of Flash memory and 20 KB of SRAM. Internally, the STM32 initializes the OLED with a sequence of I^2C commands on boot. It then continuously listens on the UART port. Upon receiving a new object name (e.g., "apple"), it clears the display and renders the new string using a bitmap font library stored in Flash memory.

- **OLED Module**
  The OLED display is a 0.96-inch ZJY096I0400BG01 monochrome screen that communicates with the STM32 via I^2C. The SCL and SDA lines are connected to PB8 and PB9 respectively. The display shows the name of the object being grasped in real time, serving as a key user interface.
- **Power Supply Module**
  This module uses an AMS1117-3.3 V regulator to convert a 5 V input from the USB into a stable 3.3 V output. The regulated voltage powers both the STM32 microcontroller and the OLED display. Input and output capacitors are included to filter out voltage ripple and ensure reliable performance.
- **Status LED Module**
  A single LED connected to GPIO pin PC13 through a 1 kΩ resistor. It is used to indicate power or operational status. The LED lights up when PC13 is pulled low.
- **Clock Module**
  An 8 MHz crystal oscillator provides the system clock for the STM32. It is connected between OSC_IN and OSC_OUT, with two 20 pF capacitors to ground and a 1 MΩ bias resistor. This ensures accurate timing for serial communication and internal processing.
- **Reset Module**
  The reset circuit includes a 10 kΩ pull-up resistor and a 0.1 µF capacitor connected to the NRST pin, along with a push-button to manually trigger system reset. It ensures proper initialization and recovery during firmware development or fault handling.
- **SWD Debugging Header**
  This module provides a 4-pin interface for Serial Wire Debug (SWD). The SWDIO and SWCLK signals are connected to PA13 and PA14 respectively, alongside 3.3 V and GND. It enables firmware uploading and real-time debugging using external tools such as ST-Link.

2.5 Path Translator

For control, we will go to inverse kinematics. After the image processing get the coordinates and orientation of the target, relative to the robotics $(\alpha, \beta, \gamma, x, y, z)$, the transformation will become:

$$
{}^{0}_{E}T = \begin{bmatrix}
c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & x \\
s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma & y \\
-s_\beta & c_\beta s_\gamma & c_\beta c_\gamma & z \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

And

$$ {}^{0}_{E}T = {}^{0}_{1}T\,{}^{1}_{2}T \cdots $$

The Denavit–Hartenberg (D-H) parameters provide a standardized method to describe the geometric relationship between adjacent links and joints of a robotic manipulator. This method is widely used to model the forward kinematics of robotic arms.

In the standard D-H convention, the relative transformation between two consecutive coordinate frames is characterized by four parameters:

- θ_i: Joint angle – the rotation about the z_{i-1} axis (variable for revolute joints).
- d_i: Link offset – the translation along the z_{i-1} axis (variable for prismatic joints).
- a_i: Link length – the translation along the x_i axis (a constant).
- α_i: Link twist – the rotation about the x_i axis (a constant).

By using these four parameters, a 4×4 homogeneous transformation matrix can be constructed to describe the pose of each link relative to its predecessor. Chaining these transformations allows for the complete forward kinematic model of the robot to be built.

For transformation ${}^{i}_{i-1}T$, we use D-H parameters to determine which is

$$ {}^{i}_{i-1}T = [Z_{i-1}][X_i] $$

$$ [Z_i] = \begin{bmatrix} \cos\vartheta_i & -\sin\vartheta_i & 0 & 0 \\ \sin\vartheta_i & \cos\vartheta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ [X_i] = \begin{bmatrix} 1 & 0 & 0 & r_{i,i+1} \\ 0 & \cos\alpha_{i,i+1} & -\sin\alpha_{i,i+1} & 0 \\ 0 & \sin\alpha_{i,i+1} & \cos\alpha_{i,i+1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The transformation matrix T is a 4×4 homogeneous transformation matrix, widely used in robotics and rigid body kinematics to describe the pose (position and orientation) of one coordinate frame relative to another.

It has the general form:

$$T = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where:

- $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix, representing orientation.
- $\mathbf{p} \in \mathbb{R}^{3\times1}$ is the position (translation) vector.
- The bottom row $[0\ 0\ 0\ 1]$ allows translation and rotation to be combined in a single matrix operation.

Back to codes itself, the getangles(x, y, z, roll, pitch, yaw) function is designed to compute the six joint angles of a robotic arm based on the desired position and orientation of the end effector, effectively performing inverse kinematics. The function first converts the input Euler angles (roll, pitch, yaw) into a rotation matrix, and then transforms it into the Denavit–Hartenberg (DH) coordinate system using a fixed rotation matrix. Using the end effector position (x, y, z) and the Z-axis direction from the rotation matrix, the function calculates the wrist center position, which is essential for solving the first three joint angles. With the help of the robot's geometric parameters and trigonometric relations, it derives q1, q2, and q3 through geometric methods. Then, it computes the rotation matrix R36 from the wrist to the end effector and extracts the remaining three joint angles q4, q5, and q6. All joint angles are finally converted from radians to degrees, and negative values are adjusted to fall within the valid actuator range. Overall, this function accurately maps a spatial pose into joint commands, serving as a core component in robotic arm control and motion planning.

However, the above codes doesn't work. We spent completely two weeks testing it and trying to find what's wrong. We also try other ways, including Moveit in ROS, and Jacobian-Based Method. The biggest problem is we can't ensure what's the relationship between points in calculation from codes and in real world. In other words, we met problem in coordinate transformation. Our visual model can successfully find the target and its position, and give the robot arm x, y and z coordinates. We tried to set x, y and z to zero separately, in order to find where (0, 0, 0) is in real world. Therefore, we don't use inverse kinetics in the end, and our robotic arm can't grasp the object from arbitrary place. It can only grasp objects from certain place.

What we use is hard code. We put 14 sample points in the paper. Each time we pull the robotic arm to the sample points one by one, and read the angles of 6 motors. The path planning portion relies on the id parameter, where each ID (from 1 to 14) corresponds to a predefined target position. These positions are represented by specific sets of six joint angles, defining a motion path for the robotic arm. These angles are precomputed using forward or inverse kinematics to ensure that the arm's trajectory is reachable and free from collisions. The motion is controlled in stages: the arm first moves above the target object, then closes the gripper to grasp it, moves to the drop-off location, and finally opens the gripper to release the object.

After each motion step, the program uses time.sleep() to ensure the action completes fully and the arm remains stable without misalignment. At the end of the function, the robotic arm returns to a default standby pose to prepare for the next operation. Overall, the main function integrates object-to-gripper mapping and ID-to-path mapping, combining kinematic modeling and sequential control to complete the full process of detection, path execution, precise grasping, and object placement. Each time visual model will detect the object, find its center, and calculate which sample point is the nearest. Then the robotic arm can go to the point directly.

2.6 Robotic Arm

We use the YahBoom DOFBOT SE Robotic Arm, which is driven by the STM32 controller. This robotic arm has 6-degree-of-freedom serial bus servo motors and is controlled by the ROS operating system. By using a computer microphone and a depth camera installed at a fixed position, we have equipped the robotic arm with visual and auditory perception capabilities. For the end effector, we design a mechanical claw with a maximum opening width of 6 cm and a maximum load of 200 g, so that it can grasp common small objects. Figure 4 shows the specifications of this robotic arm [6].

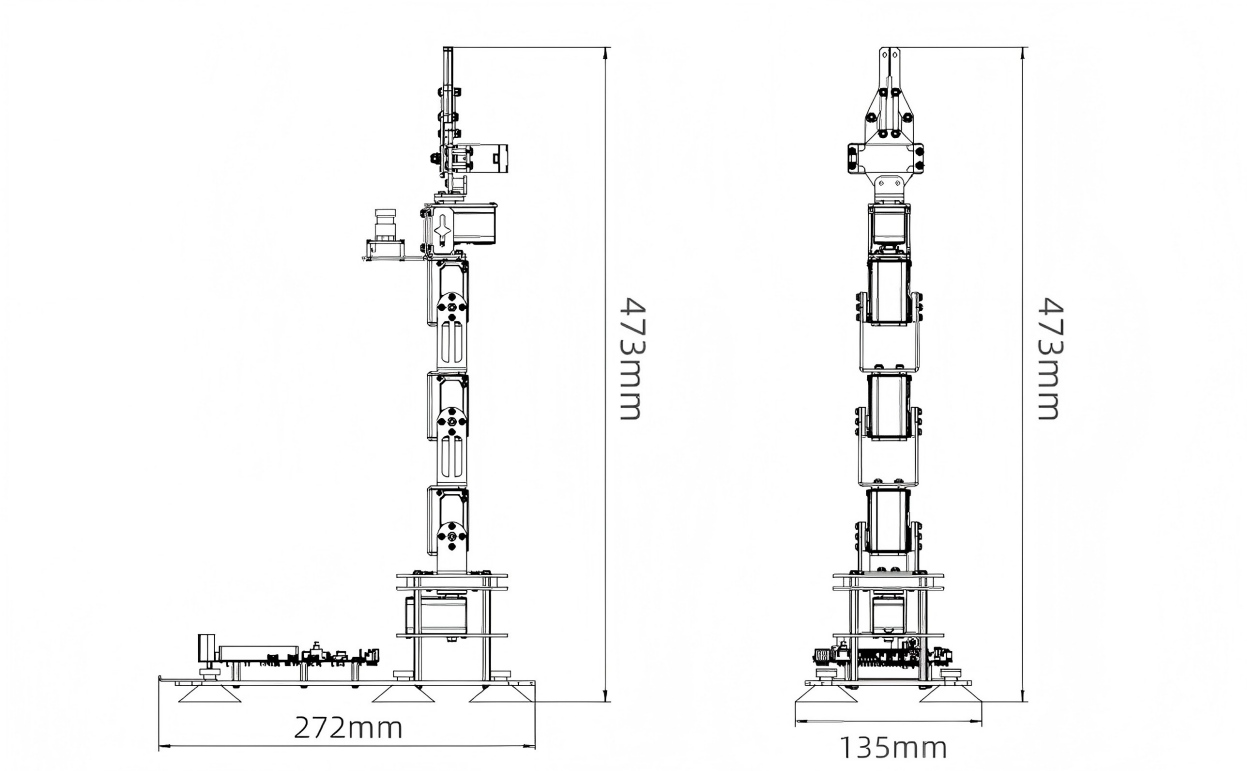


Figure 4: The specifications of the robotic arm

2.6.1 Depth Camera

In our project, we use an Intel RealSense D455i depth camera mounted in a fixed position to provide synchronized RGB and depth images to the perception system. The RGB images are processed by the YOLOE-V8L model to detect objects based on the labels extracted from the voice command. The depth images are passed to the Open-3D module to calculate the 6D spatial coordinates of the target object. The D455i depth camera is equipped with a stereo depth perception system that can provide accurate measurements at a range of up to 6 meters and has a wide field of view (approximately $90° \times 65°$) to ensure full coverage of the robot's workspace. Its built-in inertial measurement unit (IMU) further enhances depth stability during motion.

2.6.2 Microphone

We choose a built-in microphone array (Intel® Smart Sound Technology, Digital Microphone for Realtek® Audio) as the input device to capture voice commands. The captured audio is processed by a Wave2Vec 2.0 model that performs automatic speech recognition (ASR) to convert voice commands into text, allowing the system to recognize the name of the target object.

2.6.3 STM32

The STM32F103C8T6 microcontroller serves as the control hub of the robotic arm system. Based on an ARM Cortex-M3 core running at 72 MHz, it integrates 64 kB of Flash memory and 20 kB of SRAM, providing sufficient computational power for real-time control. It features multiple USARTs, PWM-compatible 16-bit timers, and GPIO ports for precise coordination of the six smart servo motors. With built-in USB, I^2C, SPI, and ADC interfaces, the STM32 manages sensor inputs, actuator commands, and peripheral communication efficiently, enabling stable and responsive operation of the robotic arm.

2.6.4 Motor

The robotic arm is driven by six DS-SY15A smart servo motors, which are responsible for executing precise and smooth movements based on the planned trajectory computed by the control algorithm. These motors operate at 6.0 – 7.4 V, deliver a rated torque of 15 kgf·cm, and reach a maximum rotation angle of 300°, allowing the arm to execute smooth and flexible movements. With a no-load speed of less than 0.24 sec/60° and metal gear construction, they ensure both responsiveness and durability. Communication is handled via UART serial protocol at 115200 bps, enabling coordinated multi-motor control through the STM32 microcontroller.

2.6.5 End Effector

Since the maximum opening width of the original robot arm end effector was too small, we redesigned the end effector, and its CAD drawing is shown in Figure 5. By laser cutting the acrylic plate, we obtained and assembled the eight parts of the end effector,

and the finished product is shown in Figure 6. The optimized end effector has a maximum opening width of 6 cm and a maximum load capacity of 200 grams, which is more suitable for grabbing various daily small objects. It is controlled by six servo motors and works in conjunction with the robot arm's motion planning system.

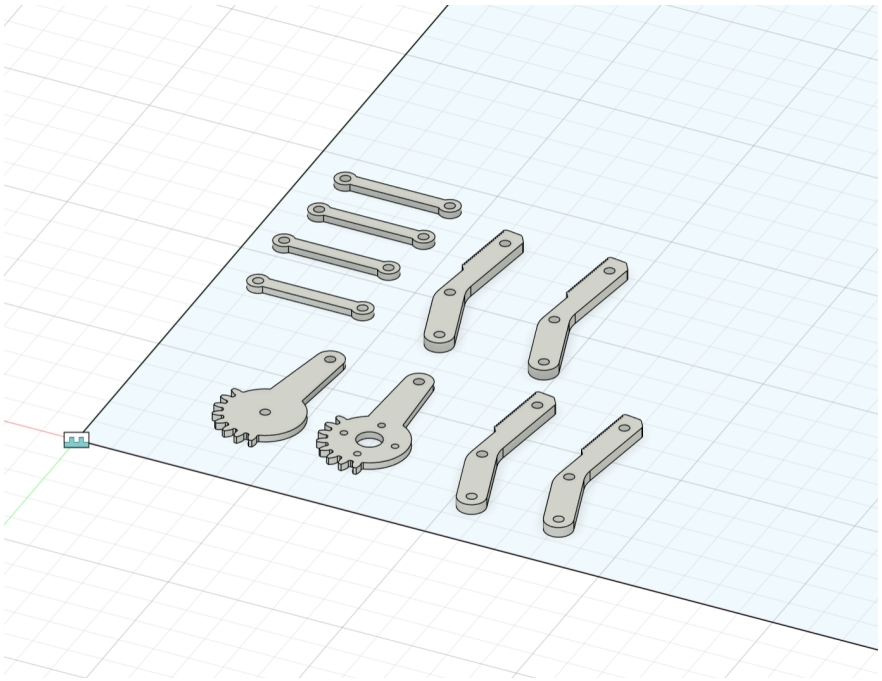


Figure 5: CAD drawing of redesigned end effector

Figure 6: Finished product of redesigned end effector

3 Verification

All the Requirement & Verification tables are included in Appendix B.

3.1 ASR (Wave2Vec 2.0)

According to Appendix B.1, ASR module has met all the requirements. To fully test its functionality, we divide the test process into twp parts. Firstly, we test it on Librispeech dataset and use Word Error Rate (WER) to evlaute the general result. For the "clean" recording, the WER is 3.4 and if we add some random noise into it, the WER is 8.6. Both are good enough to show that this ASR module can be cable of transferring speech signals to text. Another test is to speak with the microphone and check if the result meet our expectation, and we put the reuslt in the follow table:

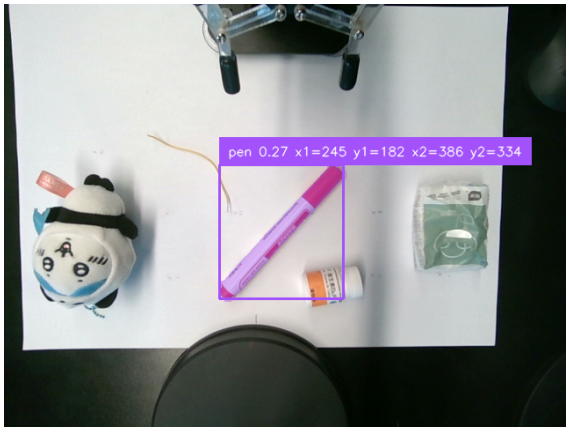
Table 2: ASR recognition examples in robotic grasping tasks

Ground Truth	**ASR Output**
Help me grasp the pen over there.	Help me graspe the pen over there.
Give me the tissue pack.	Giv me the tissue peck.
I want to have a toy.	I want to have a toy.
I need your help to get the charger.	I need your help to get the charger.
Grip the bottle gently to avoid spillage.	Grip the bottle gentley to a void spillage.

3.2 CV (YOLOE-V8L)

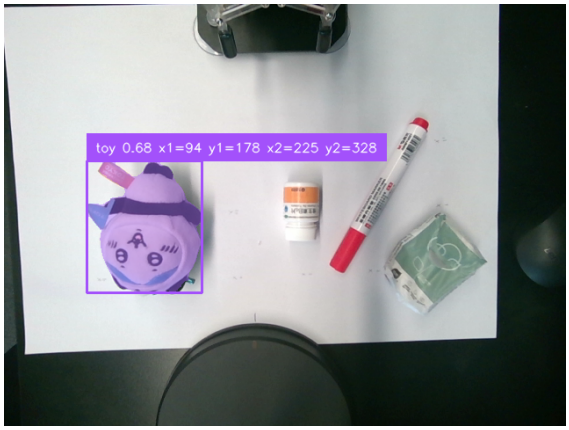
According to Appendix B.2, CV module has met all the requirements. To fully test its functionality, we follow the pseudocode 1. During each run, we save the detection result to a local folder. In Figure 7, we list some of the results. The first floating point number is the confidence level, and $[x1, y1, x2, y2]$ defines the boundary of the detection box. These images clearly show all the requirements are satisfied: it can correctly load the images from the camera; it can accurately detect target objects with a clear box according to the input label; it can obtain the frame from the depth camera whenever we pressed key `C`.

(a) Pill bottle detection



(b) Pen detection

(c) Tissue pack detection

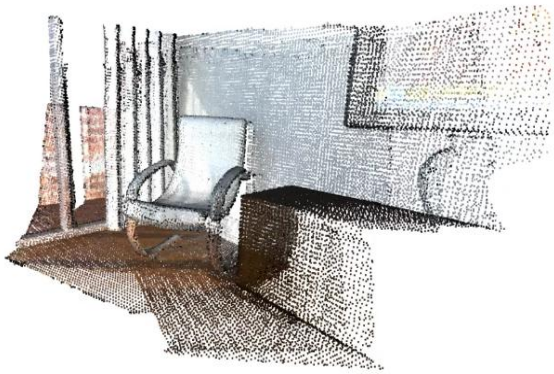


(d) Toy detection

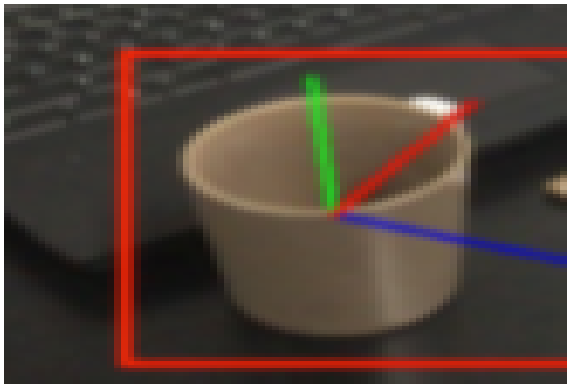
Figure 7: Working behavior of YOLOE-V8L

3.3 6D-Pose Generator (Open-3D)

According to Appendix B.3, the 6D-Pose Generator (Open-3D) module was successfully verified against all specified requirements. Using ground-truth data, the module demonstrated an average position error under 2 cm and orientation error below 5 degrees. It maintained stable performance under varying lighting, occlusion, and noise conditions, confirming its robustness. The output was consistently aligned with the camera coordinate system, with position values in meters and orientation expressed in degrees, satisfying both accuracy and unit consistency criteria. The result are as follows:



(a) the example of point cloud

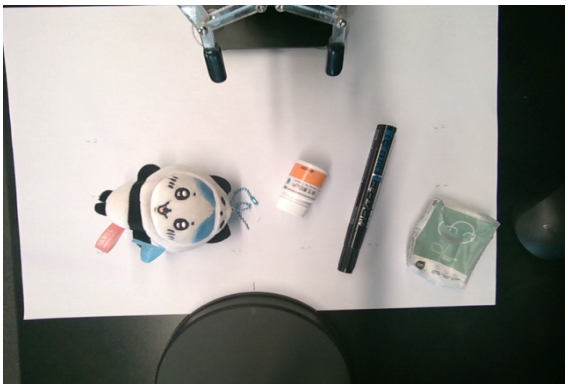


(b) the example of 6d-pose

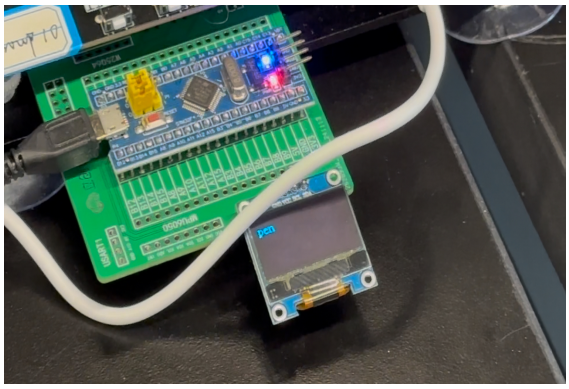
Figure 8: Working behavior of 6D-Pose Generator

3.4 PCB Display

According to Appendix B.4, PCB display module has met all the requirements. To fully test its functionality, we placed small items such as tissue bag, pen, medicine bottle, and toy on the desktop. Then, we started the program. The ASR module received the user's command "grasp a pen" and passed the label "pen" to the PCB display module. The OLED display responded within 1 second and correctly displayed the word "pen" without truncation, distortion, or corruption. The verification results are shown in Figure 9.



(a) Objects on the desktop



(b) OLED display result

Figure 9: Working behavior of PCB display

3.5 Path Translator

Since we use hard code, we can get the angles of motor directly, so we are sure the motor can move under our expectation and grab the object.

3.6 Robotic Arm

3.6.1 Depth Camera

We test the output of RGB and depth images according to Appendix B.6.1, and the depth camera module has met all the requirements.

3.6.2 Microphone

We said some sentences in noisy and quiet environments according to Appendix B.6.2, and the microphone captured them clearly.

3.6.3 STM32

According to Appendix B.6.3, we conducted two key experiments. First, we injected known test commands and sensor data from the PC (ROS system) to the STM32 via UART and confirmed that the microcontroller correctly parsed and forwarded the data with minimal delay. We recorded timestamps to evaluate response time and ensure real-time performance. Second, we introduced corrupted and delayed messages to simulate communication faults such as buffer overflows and invalid packets. The STM32 successfully identified and handled these errors without crashing, demonstrating its ability to maintain stable operation under abnormal conditions. These tests show that the STM32 module has met all the requirements.

3.6.4 Motor

According to Appendix B.6.4, we conducted three key experiments. First, we applied PWM control signals at various frequencies and loads to evaluate the motors' responsiveness and stability. The motors responded accurately with minimal overshoot or delay, confirming precise control signal interpretation. Second, we continuously monitored motor temperature, voltage, and current during operation. The readings stayed within the specified safety thresholds, and when we manually triggered overload conditions, the motors shut down as expected to prevent damage. Finally, we tested motion smoothness by executing trajectories involving abrupt changes in direction and load. The motors maintained continuous movement without jitter or stall, demonstrating robustness under dynamic control scenarios. These tests show that the motor module has met all the requirements.

3.6.5 End Effector

According to Appendix B.6.5, we used a series of test objects with varying widths (ranging from 0.5 cm to 6 cm) and varying weights (ranging from 10 g to 200 g) . The robotic arm was instructed to grasp these object, lift them, and transport them along a predefined trajectory. During the operation, the gripper successfully completed the full open-close cycle, securely held the objects without slippage, and maintained stability throughout the movement. One of the test pictures is shown in Figure 10. These tests show that the end effector module has met all the requirements.

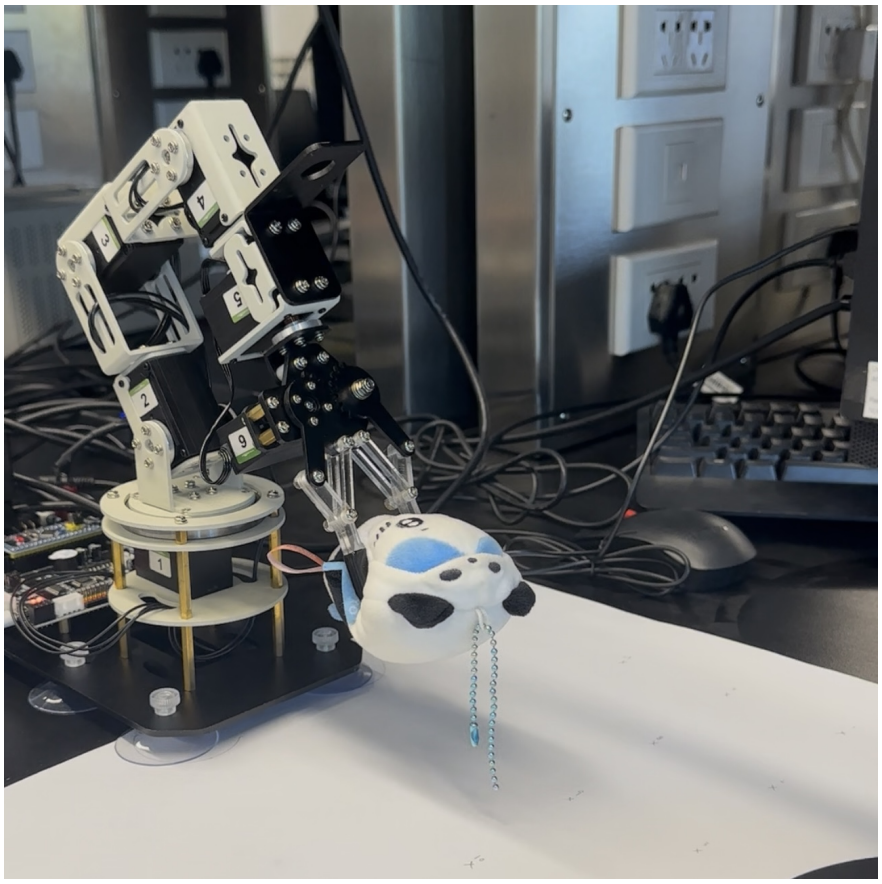


Figure 10: Working behavior of end effector

4 Cost Analysis

The total cost of the project consists of two parts: labor and hardware.

The labor cost is estimated based on the average hourly wage standard in the Electrical and Computer Engineering industry, which is 20 RMB per hour. The senior design project spanned from March to May, totaling approximately eight weeks. During this period, each team member dedicated an average of 25 hours per week to the project. Given that the team consists of four members, the total labor cost can be calculated as follows:

$$4 \times 25 \times 8 \times 20 = 16000 \text{ RMB}$$

The hardware costs include several key components. The YahBoom DOFBOT SE robotic arm was selected as the core manipulator at a cost of 1379 RMB. The Intel RealSense D455i depth camera, priced at 3050 RMB, provides spatial perception and is mounted using a 47 RMB bracket. For local visual feedback, we implemented an OLED display system consisting of a ZJY096I0400BG01 screen for 5.5 RMB and an STM32F103C6T6 microcontroller for 5.3 RMB, both integrated on a custom PCB board that cost 200 RMB. Additionally, we included a fruit model and a dice model, costing 7.5 RMB and 4 RMB respectively, for grasping testing purposes. The total hardware cost is 4698.3 RMB.

Combining the labor and hardware costs, the total expenses for the project is 20698.3 RMB, as shown in Table 3.

Table 3: Project cost breakdown

Item	**Model**	**Price (RMB)**
Robotic Arm	DOFBOT SE	1379
Depth Camera	Intel RealSense D455i	3050
Depth Camera Mounting Bracket	Logitech C1000e	47
OLED Display	ZJY096I0400BG01	5.5
STM32 Microcontroller	STM32F103C6T6	5.3
PCB Board	JLC	200
Fruit Model	–	7.5
Dice Model	–	4
Labor	–	16000
Total	–	**20698.3**

5 Schedule

Table 4: Project Schedule

Week	Tasks	Member
3/3	Complete problem definition and draft initial solution overview. Assign major modules to each member.	All members
3/10	Select STM32, microphone, and camera models. Order components.	Zixuan Zhang
	Review related work on vision-language robotic arms. Research feasible NLP and vision models.	Junzhou Fang, Junsheng Huang
	Set up development environment (ROS, PyTorch, etc.) on server and local PC.	Zixin Zhu
3/17	Design and simulate robotic arm control system in software (Gazebo or RViz).	Zixin Zhu
	Prepare NLP pretraining dataset and begin ASR pipeline development.	Junzhou Fang, Junsheng Huang
	CAD redesign and prototype the custom end effector.	Zixuan Zhang
3/24	Receive components and test camera, microphone, and STM32 communication individually.	Zixuan Zhang
	Build ROS nodes for perception and planning. Test dummy object tracking.	Zixin Zhu
	Implement keyword extraction module in NLP pipeline.	Junzhou Fang, Junsheng Huang
4/1	Assemble robotic arm and test motor control from STM32.	Zixuan Zhang
	Integrate NLP module with basic ROS interface.	Junzhou Fang, Junsheng Huang
	Build ROS serial bridge with STM32, test echo and actuator signal delay.	Zixin Zhu
4/8	Refine inverse kinematics algorithm and joint trajectory planning.	Zixin Zhu
	Finalize data collection and pretraining for NLP and object detection models.	Junzhou Fang, Junsheng Huang

Continued on next page

Table 4 – continued from previous page

Week	Tasks	Member
	Mount sensors on arm, calibrate camera position relative to base frame.	Zixuan Zhang
4/15	Finalize hardware setup: STM32, motors, camera, microphone. Connect end effector and test basic grasping.	Zixuan Zhang
	Pre-train and test NLP model on sample commands.	Junzhou Fang, Junsheng Huang
	Set up ROS environment on PC, confirm serial communication with STM32.	Zixin Zhu
4/22	Integrate microphone with ASR pipeline (Wav2Vec2.0).	Junzhou Fang, Junsheng Huang
	Verify ROS control pipeline (PC $\rightarrow$ STM32 $\rightarrow$ motor).	Zixin Zhu, Zixuan Zhang
	Conduct mechanical stress test of arm + gripper.	Zixuan Zhang
4/29	Deploy YOLOE object detection model with ROS camera stream. Test YOLOE + NLP integration for object command matching.	Junzhou Fang, Junsheng Huang
	Refine ROS path planner and integrate with grasp commands.	Zixin Zhu
5/6	Conduct system-level integration (voice $\rightarrow$ vision $\rightarrow$ grasp). Test real-time instruction cycle with common items. Begin documentation of experimental results and failures.	All members
5/13	Demo dry run & presentation rehearsal.	All members
	Debug remaining issues with ROS control loop or grasp logic.	Zixin Zhu, Zixuan Zhang
5/19	Final demo and report submission.	All members

6 Conclusion

6.1 Accomplishment

Over the course of the semester, our team successfully designed and implemented a multi-modal robotic system capable of performing object grasping tasks based on natural language instructions. We integrated key modules including automatic speech recognition (Wave2Vec 2.0), visual object detection (YOLOE-V8L), 6D pose estimation (Open-3D), real-time PCB feedback display, and a custom robotic arm controller based on STM32. Despite initial challenges with inverse kinematics, we overcame them by implementing a hard-coded pose mapping strategy based on sample points, which allowed reliable and repeatable manipulation. All system modules met or exceeded their verification criteria, confirming that our solution can accurately interpret user commands, locate objects in complex scenes, and execute precise grasping maneuvers. This demonstrates our ability to build a functional end-to-end assistive robotic platform from both software and hardware perspectives.

6.2 Impact

Our project addresses the critical need for accessible and intelligent assistive technologies, particularly for individuals with limited mobility. By enabling voice-guided robotic manipulation without requiring extensive training or environmental constraints, our system promotes autonomy and reduces reliance on caregivers. The modular architecture and cost-effective components also make it suitable for scalable deployment in home, eldercare, and rehabilitation environments. Beyond its practical use, this work serves as a proof of concept for combining modern deep learning frameworks with embedded robotics. It lays the groundwork for future development in natural language-driven robotic assistance, and highlights the potential of integrating AI with physical systems to improve quality of life.

6.3 Future Work

Future work will focus on enhancing the system's flexibility, adaptability, and user experience through three key improvements. First, we aim to implement an inverse kinematics algorithm to achieve position-free grasping. This will allow the robotic arm to compute joint configurations in real time and reach arbitrary target positions based on visual or voice input, improving its responsiveness and accuracy in dynamic environments. Second, to support size-free grasping, we plan to integrate sensors into the end effector. These sensors—such as tactile or distance sensors—will enable the system to detect object size and shape, allowing for adaptive grip control. This will improve grasping success rates and allow safe handling of delicate or irregularly shaped objects. Third, we will develop a graphical user interface (GUI) to enhance usability, especially for non-technical users such as the elderly or caregivers. The GUI will offer live video feedback, object recognition results, manual controls, and voice command logs, making system interaction more intuitive. Collectively, these enhancements will significantly improve the

system's generalization, robustness, and practicality for real-world applications such as home assistance or rehabilitation support.

6.4 Ethics and Safety

Given that our target audience primarily includes the elderly and disabled, ethics and safety are crucial aspects of our design. This section is divided into two parts to comprehensively address these concerns.

6.4.1 Ethics

This work focuses on developing a robotic arm capable of picking up diverse small objects. Such a system has practical applications in home assistance, eldercare, logistics, and disaster recovery. By enabling reliable manipulation in unstructured environments, it contributes to building assistive and autonomous systems that improve safety, reduce human workload, and enhance quality of life. As ZJUI students, we are committed to upholding ethical standards and ensuring the integrity of our project. Our team will strictly adhere to the IEEE Code of Ethics [7] and the ACM Code of Ethics [8]. We pledge to meet, but are not limited to, the following ethical responsibilities:

- Prioritize public safety, health, and well-being by adhering to ethical design principles and sustainable practices. Additionally, we are obligated to report any potential systemic risks that could lead to harm.
- Strive to benefit society by enhancing individual and collective understanding of both traditional and emerging technologies and their societal implications.
- Maintain honesty and integrity in all professional activities, strictly avoiding unethical conduct such as bribery or other illegal actions.

6.4.2 Safety

To ensure the safety of both team members and others, and to mitigate any potential hazards during the project, our team will strictly comply with the ECE 445 SAFETY GUIDELINES [9]. We will undertake, but are not limited to, the following safety measures:

- No team member is permitted to work alone in the laboratory at any time.
- All team members must complete mandatory safety training before being authorized to work in the laboratory.
- Any handling of battery charging or hazardous battery chemicals must be conducted in strict accordance with established safe usage guidelines.
- The robotic arm is designed to operate safely around humans, minimizing collision risks through motion planning and force-limited actuators. This ensures it can work in homes or care facilities without endangering users.

- The system incorporates grasp failure detection to avoid dropping or mishandling objects. This prevents accidental damage to the environment and ensures reliable object transfer.
- Adaptive force control mechanisms allow the arm to handle fragile objects without applying excessive pressure. This enhances safety when interacting with unknown or delicate items.
- Manual and automated emergency stop mechanisms are included to immediately halt motion in case of anomaly or user intervention. This provides a critical safety layer during deployment and testing.
- Safety-centric design builds user trust and promotes system adoption. By making robot actions predictable and explainable, users feel more comfortable interacting with the robot.

References

[1] B. Allen, C. Bagnati, E. Dow, *et al.*, "Report on the use of assistive robotics to aid persons with disabilities," *Undergraduate Coursework*, vol. 4, 2020. [Online]. Available: https://docs.lib.purdue.edu/ugcw/4/.

[2] P. Marti, M. Bacigalupo, and L. Giusti, "Assistive robots for the social management of health: A framework for robot design and human–robot interaction research," *International Journal of Social Robotics*, vol. 12, no. 4, pp. 681–702, 2020. DOI: 10.1007/s12369-020-00647-6. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC7223628/.

[3] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, *Wav2vec 2.0: A framework for self-supervised learning of speech representations*, 2020. arXiv: 2006.11477 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2006.11477.

[4] A. Wang, L. Liu, H. Chen, Z. Lin, J. Han, and G. Ding, *Yoloe: Real-time seeing anything*, 2025. arXiv: 2503.07465 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2503.07465.

[5] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018. DOI: 10.48550/arXiv.1801.09847. [Online]. Available: https://arxiv.org/abs/1801.09847.

[6] YahBoom, *Yahboom dofbot se robotic arm specifications*, Accessed: 2025-05-17, 2023. [Online]. Available: https://www.yahboom.com/tbdetails?id=562.

[7] IEEE. ""IEEE Code of Ethics"." (2016), [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html (visited on 05/17/2025).

[8] Association for Computing Machinery, *ACM Code of Ethics and Professional Conduct*, Accessed: 2025-05-17, 2018. [Online]. Available: https://www.acm.org/code-of-ethics.

[9] University of Illinois Urbana-Champaign, *ECE 445 Safety Guidelines*, Accessed: 2025-05-17, 2025. [Online]. Available: https://courses.grainger.illinois.edu/ece445zjui/guidelines/safety.asp.

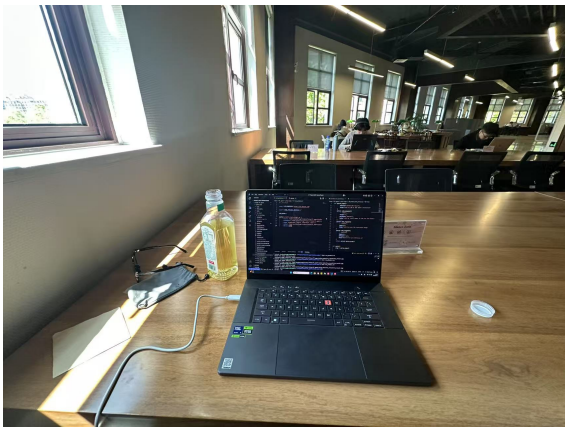
Appendix A YOLOE Analysis

YOLOE is chosen because of its high accuracy, timely classification, and generalization ability in open vocabulary. It is an advanced extension of the YOLO object detection framework that introduces strong support for open-vocabulary detection—the ability to recognize and localize objects beyond a fixed set of predefined categories. Different versions of YOLOE models can be referred to in Table 5.

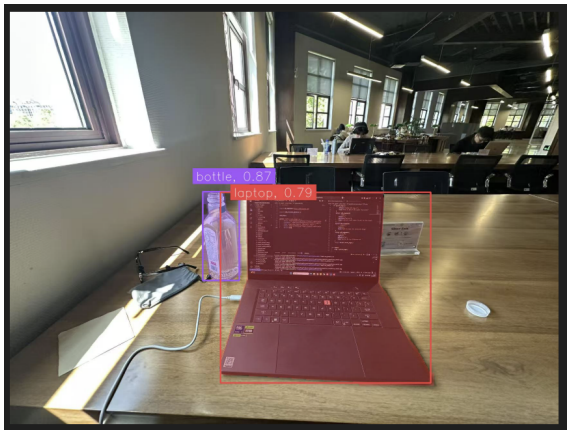
We test the performance of YOLOE-11M, YOLOE-11L, YOLOE-V8L based on a real-world daily image, with result shown in figure 11. Accordingly, we find the YOLOE-V8L has the best performance while causing acceptable resources.

Table 5: YOLOE version comparison

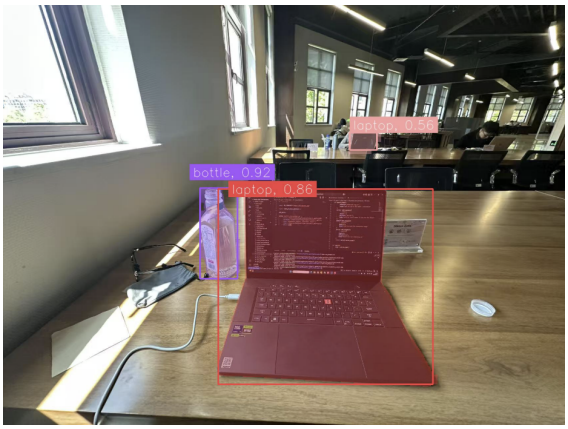
Model	**Size**	**Params**	**AP**	$\mathbf{AP_r}$	$\mathbf{AP_c}$	$\mathbf{AP_f}$
YOLOE-v8-S	640	13M	27.9 / 26.2	22.3 / 21.3	27.8 / 27.7	29.0 / 25.7
YOLOE-v8-M	640	30M	32.6 / 31.0	26.9 / 27.0	31.9 / 31.7	34.4 / 31.1
YOLOE-v8-L	640	50M	35.9 / 34.2	33.2 / 33.2	34.8 / 34.6	37.3 / 34.1
YOLOE-11-S	640	12M	27.5 / 26.3	21.4 / 22.5	26.8 / 27.1	29.3 / 26.4
YOLOE-11-M	640	27M	33.0 / 31.4	26.9 / 27.1	32.5 / 31.9	34.5 / 31.7
YOLOE-11-L	640	32M	35.2 / 33.7	29.1 / 28.1	35.0 / 34.6	36.5 / 33.8



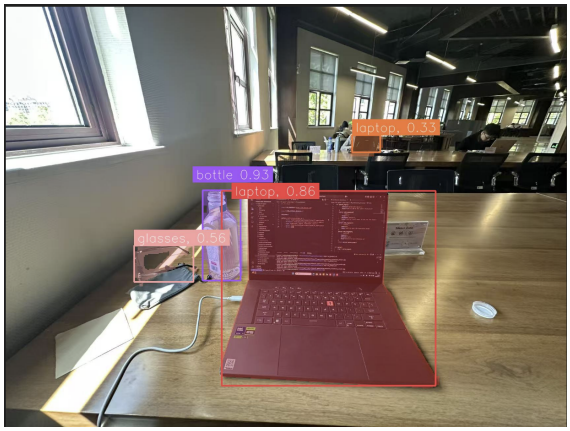
(a) Input sample



(b) YOLOE-11M



(c) YOLOE-11L



(d) YOLOE-v8L

Figure 11: Different YOLOE model comparison, with keywords: [“glasses, laptop, chairs, bottle, bottle cap”]

Appendix B Requirement & Verification Tables

B.1 ASR (Wave2Vec 2.0)

Requirement	Verification
1. Achieve accurate speech recognition and semantic understanding.	A. Design test cases with varied speech commands and noise conditions; record accuracy and compute average recognition rate. B. Test semantic label mapping accuracy under diverse natural language expressions.
2. Support real-time speech signal processing and labeling.	C. Measure time delay between receiving speech input and producing a label. D. Ensure the processing time does not exceed 200 ms.
3. Ensure adaptability to environmental noise and ambiguous inputs.	E. Evaluate performance of audio module in different environments (e.g., low noise, high noise, echo) and assess recognition stability. F. Assess semantic label module under various paraphrasing or ambiguous inputs for accuracy and robustness.

B.2 CV (YOLOE-V8L)

Requirement	Verification
1. Ensure images are correctly loaded without loss.	A. Perform a data stream test by saving images sent to both local PC and server, then manually compare the two to check for loss.
2. Ensure accurate object detection with clearly marked bounding boxes.	B. Conduct detection tests with varied backgrounds and object combinations, repeated at least 50 times to evaluate accuracy.
3. Ensure the target label (object name) is loaded correctly.	C. Log input labels during loading and compare them with expected values to verify correctness.
4. Select key frames from continuous camera input for model processing.	D. Use a frame counter to ensure that only one image is processed per instruction, validating resource-efficient frame selection.

B.3 6D-Pose Generator (Open-3D)

Requirement	Verification
1. The module shall accept a pair of aligned RGB with depth images as input and generate a 3D point cload using Open-3D. After that, the module estimates the 6D pose of the target object from the point cloud and output 3D position (x,y,z) and orientation in Euler angles (roll, pitch, yaw).	A. Use known ground-truth pose for a given RGB-D input and compare it with the estimated pose to compute position and orientation errors. B. Evaluate the module with diverse test cases featuring variations in lighting, occlusion, and perspective; record accuracy metrics.
2. The output coordinate system shall be aligned with the camera coordinate system of the depth sensor, and the position unit shall be in meters.	C. Verify that the output position is in meters and the orientation is in degrees or radians; test edge cases such as zero depth or background noise.
3. The module shall be robust to moderate noise and partial occlusion, maintaining a position error below 2 cm and orientation error below 5 degrees under test conditions.	D. Evaluate robustness by testing under different environments and check the corresponding position.

B.4 PCB Display

Requirement	Verification
1. The STM32 shall receive object names from the ROS system via UART and transmit them to the OLED display with minimal delay (less than 1 sencond).	A. Send object name strings via ROS and measure the time until correct display on the OLED using a logic analyzer or stopwatch.
2. The OLED display shall accurately render the received object name in a human-readable format without truncation, distortion, or corruption.	B. Send a variety of valid object names and verify correct on-screen rendering through visual inspection under normal operating conditions.

B.5 Path Translator

Requirement	Verification
1. Translate high-level path commands into low-level joint or actuator instructions accurately.	A. Compare the translated joint angles or control signals with expected values for known test paths. B. Test on real robot arm to ensure correct execution of generated instructions.
2. Ensure real-time performance for path computation.	C. Measure the computation time of the path translator under various path complexities to ensure responsiveness. D. Stress test the module with long or curved paths to observe latency and processing limits.
3.Maintain kinematic and motion constraints of the robotic arm during translation.	E. Validate that all generated paths respect joint limits, speed constraints, and avoid illegal positions. F. Simulate edge cases to ensure safe and predictable motion behavior.

B.6 Robotic Arm

B.6.1 Depth Camera

Requirement	Verification
1. The depth camera can accurately provide RGB and depth data for object localization in real-time.	A. Test the output of RGB and depth images under typical workspace settings to confirm 6D coordinate computation reliability.

B.6.2 Microphone

Requirement	Verification
1. The microphone can clearly capture user voice commands in real-time.	A. Voice input is tested in various noise conditions to ensure reliable audio capture.

B.6.3 STM32

Requirement	Verification
1. STM32 is able to receive, parse, and forward commands from the PC to the ROS system with minimal delay.	A. Inject known sensor data and confirm the integrity and timing of the data reported to ROS.
2.The microcontroller should handle communication errors, buffer overflows, and invalid packets correctly without system crashes.	B. Introduce corrupted or delayed messages during testing to verify error handling mechanisms.

B.6.4 Motor

Requirement	Verification
1. Motors shall respond precisely to low-level PWM or control signals with minimal overshoot or delay.	A. Apply PWM signal tests at varying frequencies and loads to evaluate control fidelity and stability.
2. Motors should operate within thermal, current, and voltage safety thresholds, and automatically shut down if thresholds are exceeded.	B. Monitor motor temperature, voltage, and current during runtime to ensure safety limits are not exceeded.
3. The motors shall maintain smooth and continuous motion during trajectory execution without jitter or stall.	C. Perform sudden changes in direction and load during trajectory execution to test for jitter, stall, or recovery failure.

B.6.5 End Effector

Requirement	Verification
1. The end effector can fully open and close to grasp objects within the rated 6 cm width.	A. The gripper is tested with objects of different sizes to verify opening and gripping performance.
2. The gripper can stably hold objects up to 200 g during movement.	B. The robotic arm is commanded to transport weighted test objects to confirm load handling and stability.