Design Report:

AISI McDonald's Emotional Companion Robot

1. General Summary

1.1. Project Background and Contexts

The AISI McDonald's Emotional Companion Robot aims to address the growing demand for AI-driven emotional support in modern society. According to a Tencent Research Institute survey, 98% of respondents expressed willingness to adopt AI companionship to alleviate social isolation. Key target groups include young adults facing work-life stress and elderly individuals experiencing loneliness due to societal role transitions. Current humanoid robots struggle to deliver nuanced facial expressions, limiting user engagement. This project seeks to bridge this gap by developing a robot capable of lifelike emotional interactions through advanced facial articulation and AI-driven voice dialogue.

1.2. Assumptions

- Use AI frameworks (SenseVoice & DeepSeek) to accurately identify the user's voice content and emotions.
- Reply to users with the voice and at the same time the Robot can make expressions that are in harmony with voice and emotions.
- Modify the existing open-source robot model to make it adapt to the design requirements, that is, to make a few basic expressions.

2. Design Concept Development

2.1. Concept Generation & Selection

The robot's design integrates three core innovations:

- **Multi - Servo Facial Articulation**: To achieve realistic facial expression simulation, the robot is equipped with 20 servos, which are precisely allocated to

key parts such as eyebrows, eyes, and mouth. Among them, 4 servos are responsible for controlling the vertical movement of the two ends of the two eyebrows, enabling the eyebrows to make rich movements such as raising, lowering, and frowning, accurately conveying emotions such as surprise, doubt, and anger. 6 servos are used to control the opening and closing of the eyelids and the rotation of the eyeballs respectively. By simulating the blinking and gaze direction changes of human eyes, the sense of reality in eye - to - eye communication is enhanced. The mouth part is driven by 10 servos working in coordination, which can realistically simulate the movements of the lips, cheeks, and jaw, vividly presenting expressions such as smiling, pouting, and laughing, providing rich details for emotional expression.

- **AI - Driven Emotion Recognition:** With the help of advanced AI models such as SenseVoice and DeepSeek, the robot has a powerful emotion recognition ability. It analyzes multi-dimensional information such as intonation, speech rate, volume, and word choice in the user's voice to accurately judge the user's current emotional state, such as happiness, sadness, anger, and anxiety. Based on these recognition results, the robot can automatically trigger matching facial expressions and voice responses, achieving more natural and smooth emotional interactions. For example, when it recognizes that the user is in a happy mood, the robot will automatically raise its eyebrows, open its eyes wide, and show a smile, and communicate with the user in a cheerful tone. If it detects that the user is in a low mood, the robot will lower its eyebrows, look dull, and respond with gentle and comforting words.
- **Modular Interaction Modes:** Considering the diverse needs of different user groups, the robot is designed with modular interaction modes. For young people who are full of vitality and love entertainment, a "fun interaction mode" is specially set up. In this mode, the robot will adopt a humorous language style, play games with users, tell jokes, and can also imitate the voices and intonations of popular movie and TV characters or celebrities, bringing a relaxed and pleasant entertainment experience to young people. For the elderly, the "soothing companionship mode" focuses more on emotional companionship and psychological comfort. The robot will communicate with the elderly at a gentle and slow pace, patiently listen to their stories and feelings, and provide services such as health - preservation knowledge and reminiscence, helping the elderly relieve loneliness and create a warm and comfortable communication atmosphere.

2.2. Design Development Milestones

- **Mechanical Prototyping:** In the initial stage of the project, the team selected an open - source 3D model on the Onshape platform as the basic mechanical structure. Through in-depth research and adaptive modification of the model, the design team skillfully planned the installation positions and movement spaces of 20 servos in the model to ensure that the servos can accurately drive the movements of various parts of the face. During the modification process, full

consideration was given to the stability and reliability of the mechanical structure. The key connection parts were strengthened, and the layout and shape of the parts were optimized to reduce friction and energy loss during the movement. After multiple simulation tests and fine - tuning, the mechanical prototype was successfully produced, laying a solid hardware foundation for subsequent facial expression simulation.

- **Software Integration:** The software team is responsible for developing the voice interaction pipeline, which covers multiple key links. First, the voice recording module is developed. High - sensitivity microphones are used to collect users' voice information and convert it into digital signals for preliminary processing. Subsequently, the signals are transmitted to the server - side, where complex processing such as emotion recognition and semantic understanding is carried out using AI frameworks such as SenseVoice and DeepSeek. Finally, corresponding voice replies are generated based on the processing results, and the text is converted into natural and smooth voices through voice synthesis technology for output. During the development process, the team continuously optimized the algorithms to improve the accuracy and fluency of voice recognition and synthesis, ensuring that the robot can accurately understand users' intentions and make appropriate responses.
- **System Testing:** To ensure that the robot's performance meets the expected standards, the system testing stage is crucial. For the response speed of the servos, the team set strict goals, requiring that the conversion delay of facial expressions be less than 0.5 seconds after the servos receive control signals. A professional testing platform was built to simulate various different expression conversion scenarios, and the response time of the servos was accurately measured and analyzed. For the accuracy of artificial intelligence, a large number of test corpora were used for multiple rounds of testing, covering voice samples of users of different ages, genders, and accents, as well as various complex emotional expression scenarios. By continuously optimizing the algorithms and adjusting the parameters, the accuracy of the robot's emotion recognition and response was gradually improved, enabling it to more accurately understand and respond to users' emotional needs.

3. Final Design Review

3.1. Overview

The AISI McDonald's Emotional Companion Robot employs a hybrid hardware-software architecture to achieve lifelike emotional interactions. The structural framework utilizes 3D-printed PETG components due to their superior thermal stability, impact resistance, and cost-effectiveness compared to ABS or PLA. PETG's flexibility also accommodates repetitive servo movements without fracturing, ensuring long-term durability.

The facial articulation system integrates 20 servo motors (6 SG90 for lightweight

movements,12 MG90S for medium load and 2 MG996R for high-torque jaw motions) controlled via an Arduino UNO and PCA9685 servo driver board. This setup enables precise coordination of eyebrow, eye, and mouth movements. The AI backbone combines SenseVoice for real-time multilingual sentiment analysis and DeepSeek for context-aware dialogue generation, hosted on a cloud server to reduce onboard computational load.

3.2. Performance Scenario

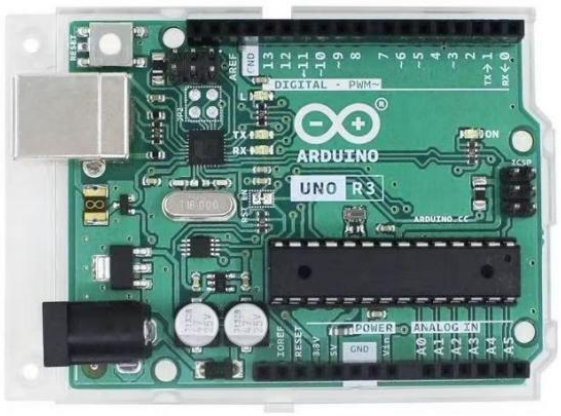
- **Emotion Recognition:** Detects user sentiment via voice tone. Multiple languages are supported. This part is mainly handled by ESP32. It collects the user's voice input in real time through the INMP441 I2S microphone and uses an energy-based speech Activity Detection (VAD) algorithm to determine whether the user is speaking. Meanwhile, the ESP32 receives the user's startup instructions and volume adjustment requests through the connected physical keys. The collected raw audio data and related control signals (such as start/end recording) will be packaged and sent to the Python server via the network for subsequent processing.
- **Voice Reply:** AI frameworks (SenseVoice & DeepSeek) are used to give the text reply. DeepSeek generates contextually appropriate responses in the user's language (currently supports Mandarin, English, Korean, and Japanese). A text-to-speech (TTS) engine synthesizes replies with dynamic intonation, synchronized with facial expressions.
- **Expression Generation:** Translates emotions into servo-actuated facial movements. The Python server sends the corresponding instruction code to the Arduino based on the sentiment judged by the LLM. After receiving instructions, Arduino controls multiple SG90, MG90S and MG996R servos connected to two PCA9685 servo driver boards to drive the robot's eyes, eyebrows, mouth and other components on its face to make preset expressions (such as neutral, happy, sad, surprised), and can also achieve timed blinking and simulate the opening and closing of the mouth when speaking according to the duration of voice playback. (e.g., raised eyebrows for happiness, downturned mouth for sadness).

3.3. System Requirements

Hardware Requirement:

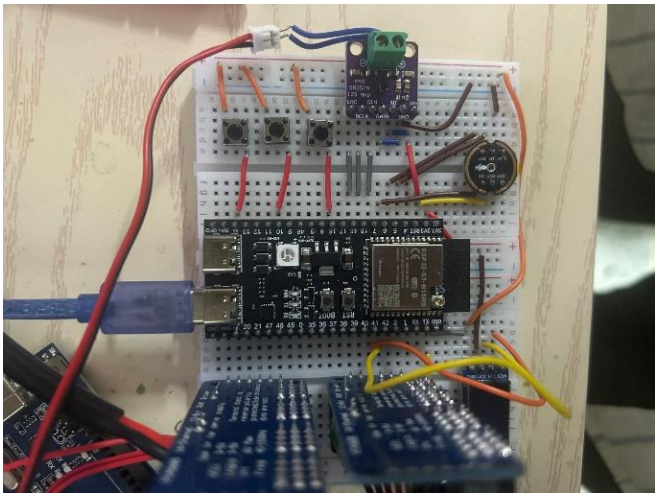
- **Arduino UNO**

Arduino UNO is a development board based on the ATmega328P microcontroller. In this project, it serves as the main controller, responsible for receiving sensor signals, processing logical instructions, and communicating with other modules (such as the PCA9685 servo driver board) through I2C or PWM signals. Its advantages include a simple and easy-to-use Arduino IDE programming environment, rich library support (such as servo control and communication protocol libraries), and scalability for connecting to modules such as ESP32 through GPIO pins. Due to the limited number of GPIOs in Arduino UNO, it is necessary to rely on the expansion module to control the multi-channel servo.



- **ESP 32 Module**

ESP32 is a dual-core microcontroller integrating Wi-Fi and Bluetooth functions, used to achieve wireless communication functions, such as connecting to a mobile phone App or a PC upper computer to receive instructions, or interacting with a cloud server through the MQTT/HTTP protocol. It serves as the perception, feedback and communication hub of the entire system in this project. This module supports multi-threaded processing and can manage communication tasks and logic control simultaneously. It is compatible with 3.3V/5V logic levels and can be directly connected to devices such as Arduino.



- **PCA9685 Servo driver board**

The PCA9685 is a 16-channel PWM driver based on the I2C protocol, designed to expand the control capability of servos. It supports 16 independent PWM outputs and multiple modules can be cascaded through address jumpers to achieve more channel control. Its 12-bit resolution (4096-level accuracy) can provide smooth servo movement, and the default frequency of 50Hz is suitable for servo control requirements. This module is compatible with 3.3V/5V logic levels, communicates with Arduino through the I2C interface, and can centrally manage multiple servo signals. It compensates for the limited number of GIPO pins of Arduino and simultaneously improves the accuracy of controlling the movement of the servo.

- **6 SG90 servos and 12 MG90S and 2 MG996R metal-gear servos**

The SG90 servo features plastic gears and a lightweight design, with a torque of approximately 1.8kg·cm. It is suitable for low-load, high-precision scenarios (the main servo used in this project). The MG90S servo can fall between SG90 and MG996R, with metal gears enhancing durability and a torque of approximately 2.2kg·cm, suitable for medium-load scenarios, mainly used in mouse in the robot. The MG996R servo is equipped with metal gears and has a torque of up to 11kg·cm, making it suitable for high-load driving (in this project, it is only used to control the jaw). The combined use of the two can balance the cost and performance requirements. Two kinds of servos used in this project are 180-degree versions which can precisely control the rotation Angle of the servos.

- **Power: 5V DC supply for servos**

The DC power supply is used for the external power supply of the PCA9685, thereby driving the servo.

Soft Requirement:

ESP32 Microcontroller Software Requirements (esp32.cpp)

Development Environment & Framework: The ESP32 firmware is developed using C/C++ within an Arduino framework-compatible environment, such as the Arduino IDE with the ESP32 core installed, or PlatformIO with VS Code. It leverages the underlying ESP-IDF and FreeRTOS for multitasking capabilities, enabling concurrent operations like audio processing, network communication, and peripheral

management.

1. Networking (WiFi.h, WiFiClient): The WiFi.h library is utilized to establish and maintain a Wi-Fi connection to the local network. A WiFiClient object is employed to create a TCP/IP socket connection with the Python backend server for transmitting audio data and control signals, and receiving synthesized speech and text replies. Custom logic handles packetization with type and length headers.

2. Audio I/O (driver/i2s.h): The ESP-IDF I2S driver (driver/i2s.h) is configured for both audio input from the INMP441 microphone and audio output to the MAX98357A amplifier. This includes setting up sample rates (16000 Hz), bit depth (16-bit), and channel formats. Custom functions like energe_vad are implemented for Voice Activity Detection on the input audio stream.

3. Peripheral Control (Adafruit_NeoPixel.h, U8g2lib.h): The Adafruit_NeoPixel.h library is used to control the RGB LED (WS2812 type on pin 48), allowing for various color codes to indicate system status (e.g., listening, processing, replying). The U8g2lib.h library drives the SSD1306 OLED display (via I2C on pins 42, 41) to show textual information, including status messages and received text replies, with support for static and scrolling text.

4. Real-Time Operations (FreeRTOS): FreeRTOS (freertos/FreeRTOS.h, task.h, queue.h, semphr.h) is integral for managing multiple tasks such as the network communication task (NetworkTaskFunction), LED control (RGB_LED), OLED display updates (u8g2_oled), and button input handling (buttom). Queues and semaphores are used for inter-task communication and resource synchronization, ensuring responsive and non-blocking operation.

Python Backend Server Software Requirements (server.py)

Development Environment & Runtime: The server-side application is developed in Python (version 3.x), intended to run on a machine accessible by the ESP32 over the network. It utilizes standard Python libraries and third-party packages for its functionalities.

1. Network Communication (socket): The socket library is used to create a TCP/IP server that listens for incoming connections from the ESP32. It handles receiving audio data and control signals, and sending back processed audio and text. It also uses sockets to communicate with the SenseVoice STT service.

2. AI Service Integration (openai, requests): The openai library is employed to interact with the DeepSeek LLM API, sending user queries (derived from STT) and a system prompt to obtain structured JSON responses containing the reply, emotion, and language. The requests library is used to make HTTP POST requests to the locally hosted GPT-SoVITS TTS API, sending text and parameters to synthesize speech.

3. Serial Communication (serial from PySerial): The serial library (PySerial) facilitates communication with the Arduino microcontroller over a USB serial port (e.g., ‘COM3’). This is used to send commands for controlling servo motors to enact facial expressions and speaking animations.

4. Audio Processing (subprocess, numpy, soundfile): The subprocess module is used to invoke ffmpeg for converting the audio output from the TTS service into the

required PCM S16LE format. numpy and soundfile are used for handling and saving the incoming audio stream from the ESP32 as a .wav file for STT processing.

5. Application Logic & State (collections.deque, json, time): Custom Python functions orchestrate the flow: esp32_connect for ESP32 connection, voice_to_text for STT, llm_process for LLM interaction, tts_process for TTS, and send_reply for responding to ESP32 collections. Deque maintains a conversation history for the LLM. The json library is used for parsing API responses and constructing requests. time is used for managing delays, particularly for synchronizing speaking animations with audio duration.

Artificial Intelligence Modules (ASR, LLM and TTS)

1. SenseVoice ASR Model: A multifunctional voice basic model that supports speech recognition (ASR), language recognition (LID), speech sentiment recognition (SER), and audio event detection (AED). It performs exceptionally well in multi-language recognition, sentiment analysis, and voice event detection. Its efficient inference architecture significantly reduces latency, supports rapid deployment and convenient fine-tuning, and is suitable for a variety of application scenarios. Evaluations show that SenseVoice outperforms mainstream open-source models in multiple tasks, especially in Chinese recognition and sentiment analysis, where it holds a leading edge. It runs in a Conda container, and the necessary libraries are in the requirements.txt document of its open-source repository. The related model is obtained in modelscope community.

2. DeepSeek LLM Model: An advanced Hybrid expert model (MoE) that adopts an innovative load balancing strategy and multi-token prediction training objectives, significantly improving training efficiency and inference performance. It is trained based on large-scale high-quality data and achieves efficient training through the collaborative optimization of algorithms and hardware. Its performance surpasses mainstream open-source models and approaches top closed-source models. Furthermore, by enhancing the reasoning ability through knowledge distillation, the processing level of complex tasks has been further improved, while maintaining stable training and a controllable output style. It is implemented by calling an API, so an API key is required.

3. GPT-SoVITS TTS Model: Possesses powerful zero-shot and few-shot speech synthesis capabilities. It can achieve instant speech conversion with just 5 seconds of sample or significantly enhance the similarity and naturalness of the timbre through 1-minute fine-tuning training. This technology supports cross-language synthesis (including Chinese, English, Japanese, Korean, Cantonese and other languages), and is equipped with a one-stop WebUI toolchain, integrating functions such as voice separation, automatic segmentation of training sets, Chinese speech recognition and text annotation, significantly lowering the threshold for users to create training data and build GPT/SoVITS models. Provide efficient and convenient speech synthesis solutions for beginners and professional developers. For our design, we adopted a

model that was fine-tuned using the voice of Chinese Kamisato Ayaka from Genshin Impact and used it as reference audio. It is a direct integration package, which the environment has already been configured, not needing to config.

Arduino Microcontroller Software Requirements (servo.cpp)

Development Environment & Platform: The Arduino firmware is developed using the Arduino IDE (version 1.8.0 or newer recommended for compatibility) and programmed in C/C++. It targets an Arduino board (e.g., Uno, Mega) capable of interfacing with PCA9685 servo driver boards via I2C.

1. PWM Driver Control (Adafruit_PWMServoDriver.h, Wire.h): The Adafruit_PWMServoDriver.h library, along with Wire.h for I2C communication, is essential for controlling the two PCA9685 16-channel PWM/servo driver boards (at I2C addresses 0x40 and 0x41). This library allows setting the PWM frequency (defined as SG90_FREQ for SG90 servos) and commanding individual servo channels.

2. Servo Angle Management (setSG90Angle): A custom function, setSG90Angle, is implemented to translate a desired angle (0-180 degrees) into the appropriate PWM pulse width (tick values) for SG90 servos (also suitable for other servo models in the design), considering their specific SG90_MIN and SG90_MAX pulse lengths. This function directs the command to the correct servo on the specified PCA9685 board.

3. Expression Implementation (netural, happiness, sadness, surprise): Specific functions are defined for each facial expression. These functions contain sequences of calls to setSG90Angle for multiple servos, setting them to predefined angles to achieve the visual representation of emotions like “neutral,” “happiness,” “sadness,” and “surprise.” A DELAY_TIME is used between servo movements to ensure smooth transitions.

4. Dynamic Animations (blink, Serial Command Handling): A blink() function and associated logic (using millis() for timing with BLINK_INTERVAL) are implemented for periodic eye blinking. The main loop() (partially shown) is responsible for listening to serial commands from the Python server. Based on received bytes, it triggers the corresponding expression functions or specific actions like eye movements or initiating/stopping a speaking animation (which would involve cyclically moving mouth servos).

3.4. Technical Research Areas

- Material

Research on materials that are easy to 3D print for the robot as a whole and research on other related hardware and consumables.

- Structure

Use SolidWorks to simulate the mechanical structure of the humanoid robot's head and FEM to analyse and optimize the structure. Conduct research on how to make robot models have the characteristics of being easy to manufacture, having strong stability, reasonable structural design and beautiful image.

- User interaction and input collection (Voice)

- Multi-device communication and network transmission
- Intelligent processing and decision generation (Server side)
- Feedback output (Voice)
- Expression display
- System control and task management

3.5. Materials

- **Structural: PETG (3D-printed)**

PETG is a high-performance 3D printing material used to build the frames, joints or shells of mechanical structures. It combines high strength and flexibility, making it suitable for dynamic components. It has a wide temperature resistance range (-40°C to 80°C), is resistant to ultraviolet rays and has a low printing shrinkage rate (about 0.2%). The finished product has a smooth surface and high precision, making it suitable for complex hollowed-out or suspended structure designs. PETG is printed by FDM technology and is an easy-to-peel support material, meeting the long-term stable operation requirements of this project.

- **Connection of electronic components: DuPont wires**

Dupont cables connect electronic modules (such as Arduino, ESP32, and servo driver boards) through standardized interfaces (male/female terminals) to achieve signal and power transmission. It can flexibly adapt to a 2.54mm pin spacing, support plug-and-play debugging, and quickly adjust the circuit layout without soldering. Color classification (red - power, black - ground, yellow/green - signal) reduces the risk of wiring errors and is suitable for temporary prototype construction, sensor signal transmission, or I2C communication connection between the servo driver board and the main control.

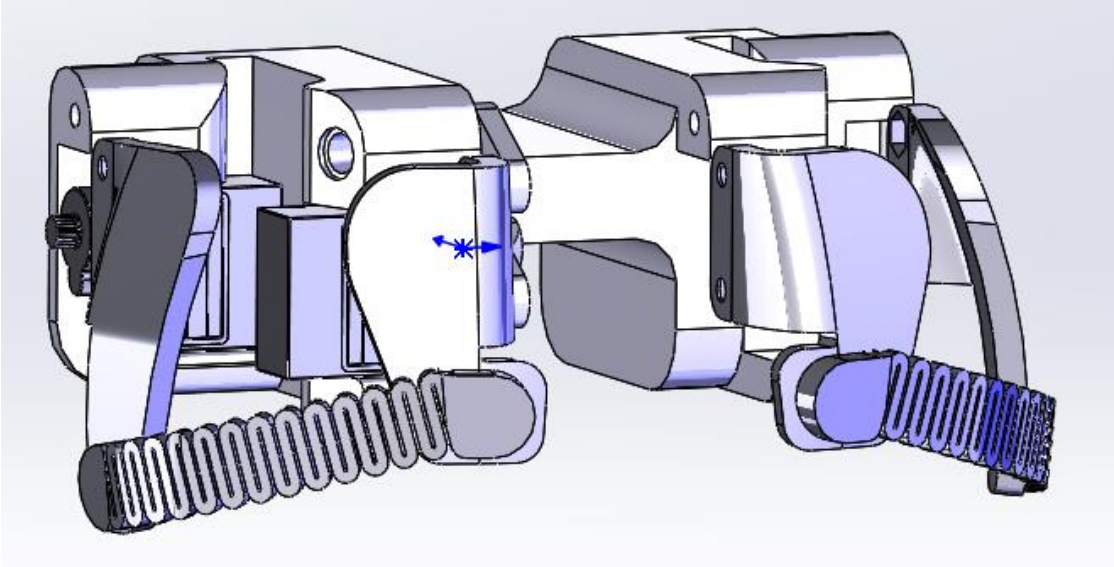
- **Assembly materials: Hot melt adhesive, Plain screws, Shoulder screws and Heat-set nuts**

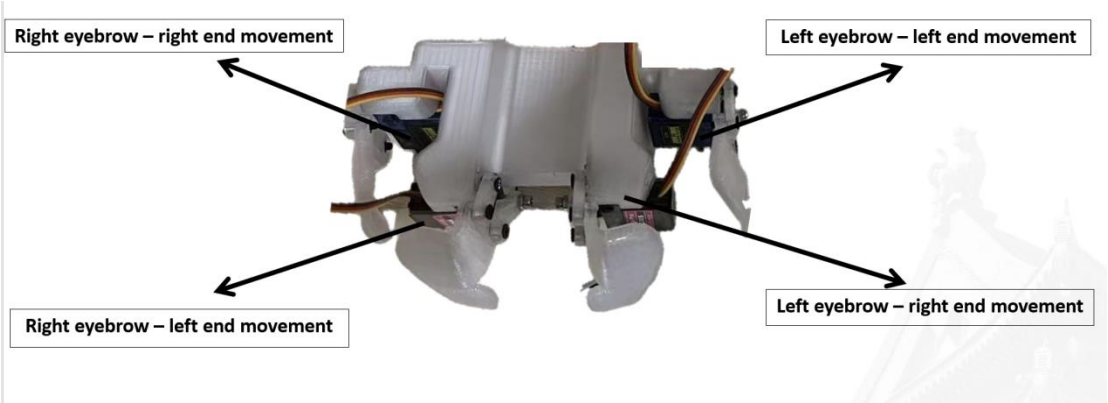
Assembly materials ensure structural reliability through bonding and mechanical fixation: Hot melt adhesives are used to quickly bond non-metallic components (such as PETG and PCB), fill gaps and insulate, making them suitable for temporarily fixing cables or lightweight components. Common screws (M2-M4) are used to achieve the splicing of detachable shells or the fixation of circuit boards. The shoulder screw positions the rotating joint through a threadless optical shaft to reduce frictional resistance. Hot-pressed nuts are embedded in 3D printed holes to form high-strength threads, preventing plastic slippage. They are suitable for repeatedly disassembled and assembled force-bearing points (such as servo mounts), and can be combined with screws to achieve modular part assembly.

3.6. Structural Design

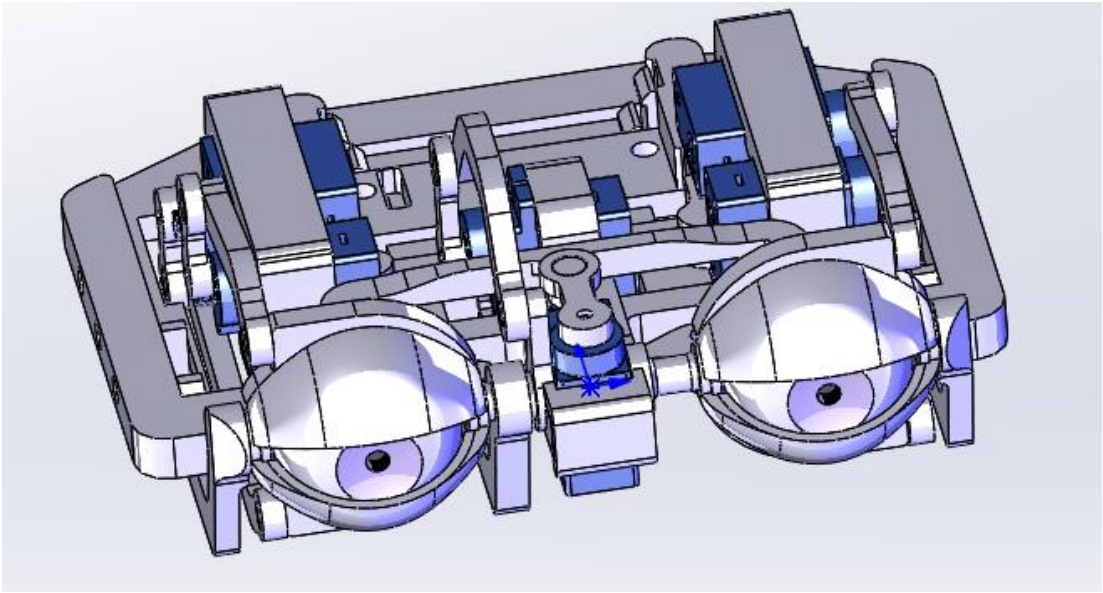
The structure of this robot is designed with the goal of fully simulating the human head. It adopts a modular design and is mainly composed of three parts: eyebrows, eyes, chin and frame.

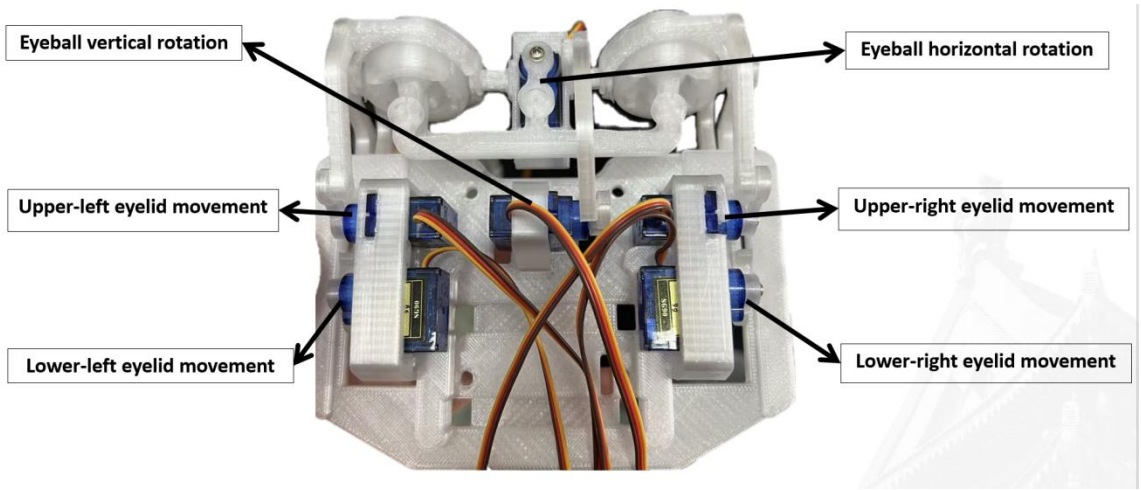
- **Eyebrows:** 4 servos respectively enable the two ends of two brows' vertical adjustments.



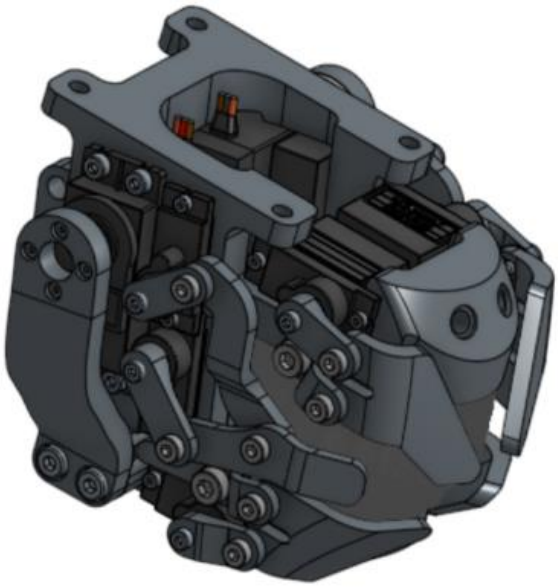


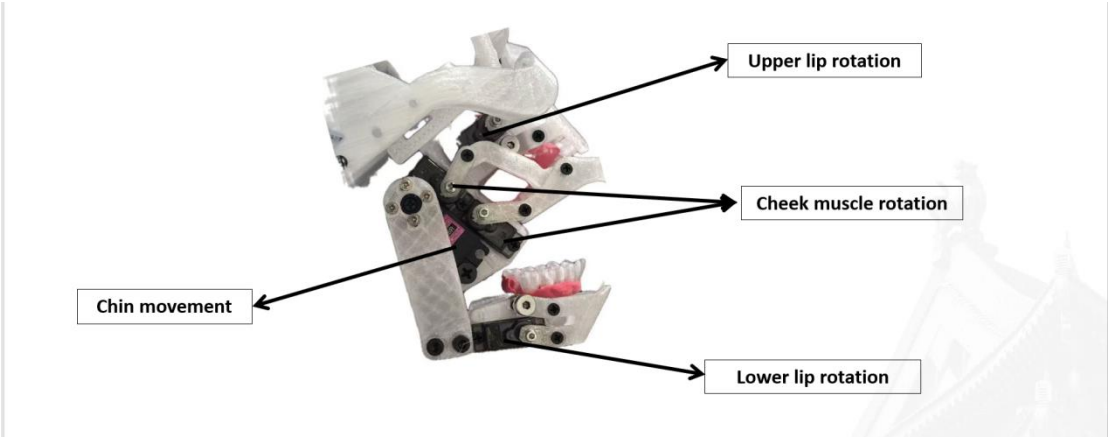
- **Eyes:** 4 servos control the opening and closing of the eyelid's and 2 servos control the eyeball's rotation.





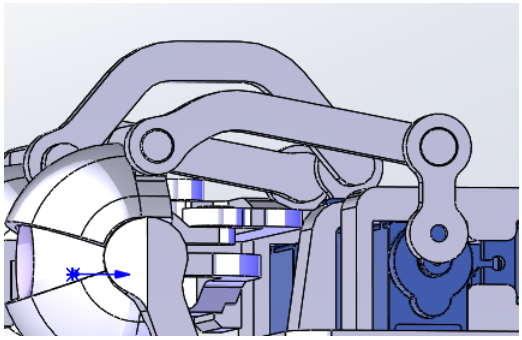
- **Mouth:** 10 servos simulate lip, cheek, and jaw movements.

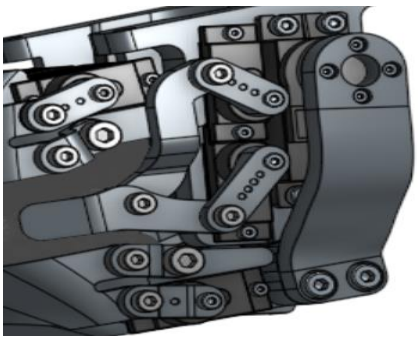




- **Frame:** Assemble and fix the remaining three parts as the main framework.

- **Movement mode of the servo driving components**: Mainly uses a linkage mechanism. The components are mainly connected by shoulder screws, and a small part is connected by a circular ring that fits with the cylinder, thus enabling each part to rotate freely.





3.7. Design for other aspects of System Performance

- **Voice interaction module**

1. User interaction and input collection

The user interaction of this system revolves around physical buttons, OLED status prompts and RGB LED color feedback: Users trigger voice input through buttonStart, and the volume buttons adjust the global volume in real time; After startup, the red LED indicates standby, and the OLED shows the activated status. When voice is detected, the LED turns green and displays “Listening”, and then the audio is continuously uploaded. During the waiting period for the response, the LED blue flash cooperated with “The girl is praying...” Prompt the processing progress; After

the reply arrives, the LED turns purple and scrolls to display the text, while the voice is played simultaneously, and the volume is adjusted adaptively. If the voice is not detected within the time limit or there is a long period of silence, the system will automatically reset to the idle state of the orange LED. The entire process forms a closed-loop interaction through LED color flow, OLED multi-mode text (static/scrolling), and physical buttons, visually presenting the status of each stage of recording, transmission, processing, and playback.

Audio processing is one of the core functions of ESP32. The i2s_begin function is responsible for initializing two I2S channels: one for the audio input (RX) of the INMP441 microphone, and the other for the audio output (TX) of the MAX98357A amplifier. Both are configured in the main mode, with a 16kHz sampling rate, 16-bit accuracy, and left channel. Audio data is transmitted through DMA, and the buffer size is defined as BUFFER_SIZE. The energe_vad function implements a simple energy-based speech activity detection algorithm. It determines whether someone is speaking by calculating the average energy of the audio frame and comparing it with VAD_THRESHOLD. Due to space limitations, we only present the pseudocode of the core logic.

```
// ESP32 pseudocode
Procedure ESP32_Handle_User_Input_And_Audio_Capture:
  Initialize I2S_Microphone
  Initialize Physical_Buttons (Start_Button, Volume_Buttons)
  Current_System_State = IDLE

  Loop Forever:
    Read Start_Button_State
    Read Volume_Button_States

    IF Volume_Up_Button_Pressed:
      Increase_System_Volume()
    IF Volume_Down_Button_Pressed:
      Decrease_System_Volume()

    IF Start_Button_Pressed AND Current_System_State is IDLE:
      Current_System_State = LISTENING
      Update_LED_Indicator(LISTENING_COLOR)
      Update_OLED_Display("Hearing...")
      Send_Signal_To_Server(START_VOICE_RECEIVE)
      Clear_Audio_Buffer()
      Last_Speech_Activity_Time = Current_Time()
      Recording_Start_Time = Current_Time()

    IF Current_System_State is LISTENING:
      Read Audio_Chunk from I2S_Microphone
      Is_Speech_Detected = Perform_Energy_VAD(Audio_Chunk)
```

```
    IF Is_Speech_Detected:
      Add Audio_Chunk to Audio_Buffer
      Send_Audio_Chunk_To_Server(Audio_Chunk)
      Last_Speech_Activity_Time = Current_Time()
    ELSE (No Speech Detected):
      IF (Current_Time - Last_Speech_Activity_Time > MAX_SILENCE_INTERVAL) OR \
         (Current_Time - Recording_Start_Time > MAX_RECORDING_DURATION):
        Send_Signal_To_Server(STOP_VOICE_RECEIVE)
        Current_System_State = PROCESSING
        Update_LED_Indicator(PROCESSING_COLOR)
        Update_OLED_Display("Processing...")
  Delay_Briefly()
```

2. Multi-device communication and network transmission

The ESP32 first connects to the designated WiFi network, and then establishes a TCP client connection with the Python server for uploading audio data and control signaling, as well as receiving processed voice and text results. The Python server, as a TCP server, receives connections from ESP32. Meanwhile, it also acts as a client to connect to the SenseVoice ASR service via TCP and interacts with the DeepSeek LLM API and the local GPT-SoVITS TTS service through HTTP POST requests. Meanwhile, it communicate with the Arduino servo control board through the USB serial port. Arduino receives instructions from the Python server via the serial port.

```
// ESP32 pseudocode
Procedure ESP32_Network_Communication:
  Function Connect_To_WiFi(SSID, Password): ...
  Function Connect_To_TCPServer(Host, Port): ... Returns Client_Socket

  Function Send_Data_To_Server(Client_Socket, DataType, Data_Payload):
    IF DataType is AUDIO_DATA:
      Packet = Create_Audio_Packet(Data_Payload)
    ELSE IF DataType is CONTROL_SIGNAL:
      Packet = Create_Signal_Packet(Data_Payload)
    Client_Socket.Send(Packet)

  Function Receive_Data_From_Server(Client_Socket):
    Voice_Response_Length = Client_Socket.Read_Int32()
    Voice_Response_Data = Client_Socket.Read_Bytes(Voice_Response_Length)
    Text_Response_Length = Client_Socket.Read_Int32()
    Text_Response_Data = Client_Socket.Read_Bytes(Text_Response_Length)
    Return Voice_Response_Data, Text_Response_Data

// Python server pseudocode
```

```
Procedure Python_Server_Communication_Hub:
  Function Setup_TCP_Server(Host, Port): ... Returns Server_Socket
  Function Accept_ESP32_Connection(Server_Socket): ... Returns ESP32_Client_Socket
  Function Connect_To_SenseVoice_ASR(): ... Returns ASR_Client_Socket
  Function Connect_To_Arduino_Serial(Port, Baudrate): ... Returns Arduino_Serial_Port

  Function Send_To_ASR(ASR_Client_Socket, Command): ...
  Function Receive_From_ASR(ASR_Client_Socket): ... Returns Text

  Function Call_LLM_API(Text_Input, History): ... Returns LLM_Response_JSON (via HTTP POST)
  Function Call_TTS_API(Text_To_Speak, Language): ... Returns Audio_Data (via HTTP POST)

  Function Send_To_ESP32(ESP32_Client_Socket, Audio_Data, Text_Data):
    ESP32_Client_Socket.Send(Length(Audio_Data))
    ESP32_Client_Socket.Send(Audio_Data)
    ESP32_Client_Socket.Send(Length(Text_Data))
    ESP32_Client_Socket.Send(Text_Data)

  Function Send_To_Arduino(Arduino_Serial_Port, Command_Byte):
    Arduino_Serial_Port.Write(Command_Byte)

// Arduino pseudocode
Procedure Arduino_Serial_Communication:
  Function Setup_Serial(Baudrate): ...
  Function Read_Serial_Command():
    IF Serial.Available():
      Return Serial.Read_Byte()
    Return NO_COMMAND
```

3. Intelligent processing and decision generation (server side)

The Python server is the intelligent core of the entire system. After receiving the audio data sent by ESP32, it saves it as a temporary WAV file and calls the SenseVoice ASR service to convert the speech into text. Next, this text (combined with the context history) is sent to the DeepSeek LLM API. The LLM will generate the reply text, determine the emotions that the reply should carry (such as happiness, sadness, etc.) and the language type of the reply. This process completes the key steps from user input to machine understanding and response content generation.

```
// Python server pseudocode
Global Chat_History_Deque

Procedure Server_Process_User_Input:
  // Assumes audio data is received and stored in `User_Audio_File_Path`
  // 1. Speech-to-Text (ASR)
```

```
  Send_ASR_Command(VOICE_TO_TEXT_COMMAND) // to SenseVoice via TCP
  User_Transcribed_Text = Receive_ASR_Result()

  // 2. Language Model Processing (LLM)
  LLM_Messages = Prepare_LLM_Messages_With_History(System_Prompt, Chat_History_Deque,
User_Transcribed_Text)
  LLM_Response_String = Call_DeepSeek_LLM_API(LLM_Messages) // HTTP POST
  Parsed_LLM_Output = Parse_JSON(LLM_Response_String) // Extracts "reply", "emotion",
"language"

  Bot_Reply_Text = Parsed_LLM_Output.reply
  Bot_Emotion = Parsed_LLM_Output.emotion
  Bot_Language = Parsed_LLM_Output.language

  // 3. Update Chat History
  Add_To_History(Chat_History_Deque, User_Transcribed_Text, Bot_Reply_Text)

  Return Bot_Reply_Text, Bot_Emotion, Bot_Language
```

4. Feedback output (voice)

The server invokes the TTS service based on the output of the LLM to synthesize the reply text into speech. The synthesized voice data and text responses are sent back to ESP32 via TCP. After receiving the message, the ESP32 drives the MAX98357A speaker through the I2S interface to play voice and displays text information on the OLED screen (with scrolling support).

```
// Python server pseudocode
Procedure Server_Generate_And_Deliver_Voice_Response(Bot_Reply_Text, Bot_Language):
  // Text-to-Speech (TTS)
  Synthesized_Audio_Raw = Call_GPT_SoVITS_TTS_API(Bot_Reply_Text, Bot_Language) // HTTP POST
  Synthesized_Audio_PCM = Convert_Audio_To_PCM_S16LE(Synthesized_Audio_Raw,
Target_Sample_Rate=16000) // using FFmpeg

  // Send to ESP32
  Send_To_ESP32(ESP32_Client_Socket, Synthesized_Audio_PCM, Bot_Reply_Text)

// ESP32 pseudocode
Procedure ESP32_Output_Feedback(Received_Audio_PCM, Received_Text):
  Update_LED_Indicator(RESPONDING_COLOR)
  Update_OLED_Display_Scrolling(Received_Text)
  Play_Audio_On_Speaker(Received_Audio_PCM, System_Volume) // Via I2S to MAX98357A
  Update_LED_Indicator(IDLE_COLOR) // After response
```

- **Expressions display**

The Arduino side (servo.cpp) is responsible for the specific facial expression execution. The server parses the emotions contained in the strings returned from LLM and maps them to hexadecimal codes (e.g., ‘happiness’ corresponds to 0x11) using the emotion_dir dictionary, and then sends the emotion commands to the Arduino to trigger the corresponding facial expressions, which are controlled by the Adafruit_PWMServoDriver library. The code uses the Adafruit_PWMServoDriver library to control two PCA9685 servo driver boards (addresses 0x40 and 0x41 respectively) to drive up to 18 SG90 servos and 2 MG996R servos. setSG90Angle function maps the input angle (0-180 degrees) to the pulse width required by the SG90 servos and converts it to the tick value of the PCA9685, and drives the specified servos to the target location. The setup function initialises the serial communication and the two PCA9685 driver boards, and sets the PWM frequency to 50 Hz. The code has predefined a number of expression functions, such as netural (neutral), happiness, sadness and surprise. Each expression function internally calls a series of setSG90Angle to set the angle of each helm (e.g., eyeball, eyelid, eyebrow, and mouth components) to form a specific facial expression.

```
// Python server pseudocode
Procedure Server_Generate_And_Deliver_Emotion_Response(Bot_Emotion):
// Control Arduino Expressions
  Emotion_Command = Map_Emotion_String_To_Byte(Bot_Emotion) // e.g., "happiness" -> 0x11
  Send_To_Arduino(Arduino_Serial_Port, Emotion_Command)

// Arduino pseudocode
Procedure Execute_Facial_Expression(Current_Emotion_State)
  setSG90Angle(TargetServo, TargetAngle) // Send specific PWM signal to the servo

Procedure Arduino_Control_Servos_And_Expressions:
  PCA9685_PWM_Init(SG90_FREQUENCY, PCA9685_ADDRESS)
  Current_Emotion_State = NEUTRAL

  Loop Forever:
    Received_Command = Read_Serial_Command()

    IF Received_Command is an Emotion_Command (e.g., 0x10-0x13):
      Current_Emotion_State = Map_Command_To_Emotion(Received_Command)
      Execute_Facial_Expression(Current_Emotion_State)

    IF Is_Time_To_Blink():
      Perform_Blink_Animation(Current_Emotion_State) // Close and open eyelid servos
      Reset_Blink_Timer()

    Delay_Briefly()
```

- **The combination of voice and expressions display**

After the expression display is complete, the server sends a 0x21 instruction to the Arduino to start playing the mouth animation. The server time.sleep waits for a period of time equal to the length of the synthesised voice to simulate the voice playback process. The above process occurs accompanied by the ESP32 driving the MAX98357A speaker to play the voice through the I2S interface. When the playback is finished, it sends 0x22 to the Arduino to stop the mouth animation, and then sends 0x10 to make the Arduino restore the facial expression to the 'neutral' state.

```
// Python server pseudocode
Procedure Server_Generate_And_Deliver_Speaking_Response():
  // Control Arduino Speaking Animation
  Send_To_Arduino(Arduino_Serial_Port, SPEAKING_START_COMMAND) // e.g., 0x21
  Speech_Duration_Seconds = Calculate_Duration(Synthesized_Audio_PCM)
  Wait(Speech_Duration_Seconds)
  Send_To_Arduino(Arduino_Serial_Port, SPEAKING_STOP_COMMAND) // e.g., 0x22
  Send_To_Arduino(Arduino_Serial_Port, NEUTRAL_EXPRESSION_COMMAND) // Revert to neutral
after speaking

// Arduino pseudocode
Procedure Arduino_Control_Servos_And_Expressions:
  Is_Speaking_Active = false

  Loop Forever:
    Received_Command = Read_Serial_Command()

    IF Received_Command is SPEAKING_START_COMMAND (0x21):
      Is_Speaking_Active = true
    ELSE IF Received_Command is SPEAKING_STOP_COMMAND (0x22):
      Is_Speaking_Active = false
      Set_Mouth_To_Default_For_Emotion(Current_Emotion_State)

    IF Is_Speaking_Active:
      Animate_Mouth_Open_Close() // Control mouth servo(s)
    Delay_Briefly()
```

5. System control and task management

The ESP32 utilizes the FreeRTOS operating system to create multiple concurrent tasks to handle network communication (data transmission and reception), audio I/O (microphone acquisition and speaker playback), RGB LED status updates, OLED screen display, and physical key detection respectively. The tasks communicate and synchronize resources through queues and mutex locks. Temporary data storage is implemented by the traditional manual memory management method of the C language (i.e., malloc() and free()). The Python server sequentially coordinates the

invocation of various AI services (ASR, LLM, TTS) and data interaction with ESP32 and Arduino through the main program flow. Arduino constantly detects serial port instructions in its loop() function, updates the servo state based on the received instructions, and independently manages periodic actions such as timed blinking.

```
// ESP32 pseudocode (Conceptual FreeRTOS Task Management)
Procedure ESP32_Main_System_Control:
  Initialize_Hardware_And_Peripherals()
  Create_Inter_Task_Queues() // For Network I/O, LED control, OLED display
  Create_Mutexes() // For shared resources like volume

  Task_Button_Monitor:
    Loop: Detect_Button_Presses(); Post_Events_To_Queues_Or_Modify_State_Variables();

  Task_Audio_Input_And_VAD: // (As detailed in Topic 1)
    Loop: Capture_Audio(); Perform_VAD(); If_Speech_Send_To_Network_Task_Queue();

  Task_Network_Communication:
    Loop:
      IF Data_In_Outgoing_Network_Queue: Send_To_Server();
      IF Data_Received_From_Server: Parse_Response();
Post_To_Audio_Output_Queue_And_OLED_Queue();

  Task_Audio_Output:
    Loop: IF Audio_Data_In_Queue: Play_Audio_On_Speaker();

  Task_OLED_Display:
    Loop: IF Text_Data_In_Queue: Update_OLED();

  Task_RGB_LED_Control:
    Loop: IF LED_State_Change_In_Queue: Update_LED_Color();

  Start_FreeRTOS_Scheduler()

// Python server pseudocode (Sequential Control Flow)
Procedure Python_Server_Orchestration:
  Initialize_All_Connections (ESP32_Server, ASR_Client, Arduino_Serial)
  Initialize_LLM_Client()
  Initialize_Chat_History()

  Loop Forever:
    Wait_For_Audio_From_ESP32() // Blocks until ESP32 sends full audio or segments
    User_Audio_Data = Receive_Full_Audio_From_ESP32()
```

```
    Bot_Reply, Emotion, Language = Server_Process_User_Input(User_Audio_Data) // Calls ASR,
LLM

    Server_Generate_And_Deliver_Response(Bot_Reply, Emotion, Language) // Calls TTS, sends
to ESP32 & Arduino

// Arduino pseudocode (Loop-based Control)
Procedure Arduino_Main_Control_Loop:
  Setup_Servos_And_Serial()
  Set_Initial_Expression(NEUTRAL)
  Last_Blink_Time = Current_Time()

  Loop Forever:
    Process_Serial_Commands_If_Available() // Updates emotion state, speaking flag
    Update_Servos_For_Current_Emotion_State()
    Perform_Timed_Blink_If_Due(Last_Blink_Time)
    Perform_Speaking_Animation_If_Active()
    Delay_Small_Amount()
```

3.8. Design for Manufacture

- Modular assembly reduces production complexity

The whole structure is composed of four parts: eyebrows, eyes, mouth and skull frame. The components of each part can be printed and assembled through 3D printing, thus achieving modular manufacturing.

- Low cost

The SG90 is selected as the main servo to reduce the cost of the drive system. At the same time, PETG material is selected as the whole to ensure low cost while guaranteeing the lightweight and low cost of the system.

3.9. Testing

- Voice collection, processing and intelligent response

Record the user's voice through the microphone to test the noise suppression ability and signal clarity, verify the accuracy of speech recognition (ASR) (supporting a mixture of Chinese and English and emotional intonation), and evaluate the intention classification ability of the natural language processing (NLP) module; Combine TTS to generate voice responses, test the response delay (target < 500ms) and the matching degree with the content, and ensure the logical coherence of the dialogue.

- Servo calibration using Arduino-based PWM controls

Use Arduino to send PWM signals to the PCA9685 to calibrate the initial Angle of the servo, and test the range and smoothness of the basic actions (opening the mouth, blinking, eye movement, and lip movement). The multi-servo motor is driven by preset parameter combinations to achieve three expressions: happiness (upturned corners of the mouth + squinting eyes), surprise (open eyes + open mouth), and

sadness (downturned corners of the mouth + drooping eyes), optimizing the synchronization error of the movements (target < 50ms) and smoothness.

- The combination of voice function and expression display

Verify the reliability of triggering corresponding expressions based on the classification results of voice keywords or emotions (for example, “happy” activates the happy expression), and test the parallel execution ability of voice processing and servo control; Measure the end-to-end delay (target < 1 second), optimize the communication protocol (serial port/UDP), and check the emotional consistency between expressions and voice content (such as sad voice combined with slow blinking).

- User trials to assess emotional resonance.

Observe the natural reactions of users through standardized dialogue tasks, and collect subjective ratings (authenticity of expressions, emotional communication ability) and objective data (micro-expressions, changes in heart rate); Evaluate the emotional matching degree (≥ 4 points proportion), delay tolerance (threshold < 1.5 seconds), and interaction naturalness (whether it causes confusion or discomfort), and quantify the emotional expression ability of the system.

3.10. Resource Use

- CAD module
- SenseVoice ASR: https://github.com/FunAudioLLM/SenseVoice
- DeepSeek LLM API: https://platform.deepseek.com
- GPT-SoVITS TTS: https://github.com/RVC-Boss/GPT-SoVITS

4. Suggested Future Developments

I. Enhance Hardware Performance and Appearance

(i) Optimize the mechanical structure

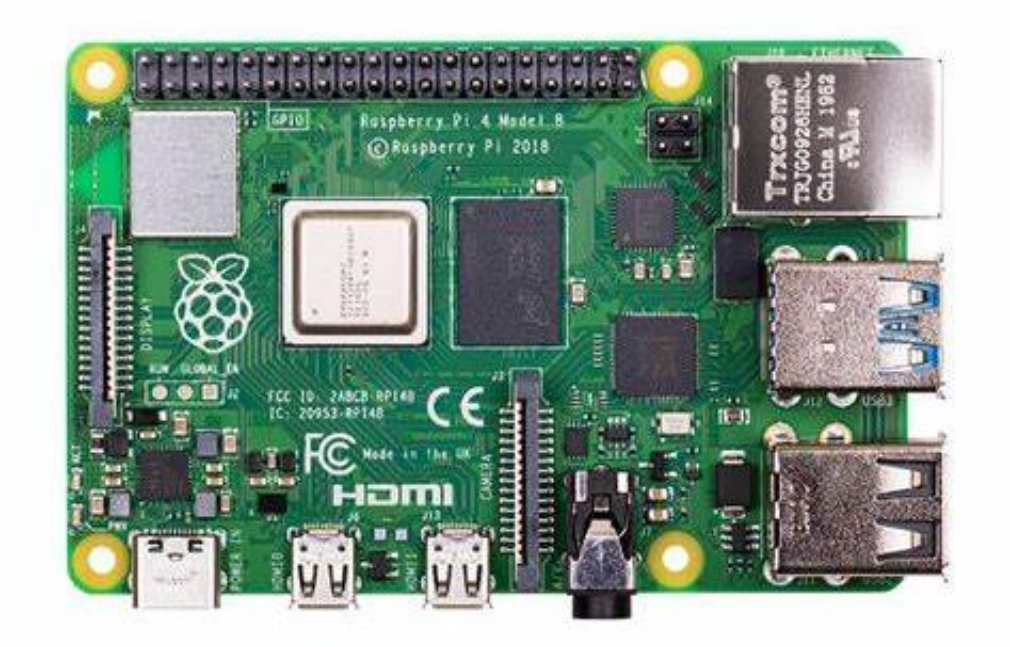
- The existing servo has limitations in expression switching speed and precision, and a high-performance servo with greater torque and faster response speed can be used, such as a metal gear servo, to reduce the expression delay and make the expression change more smooth and natural.
- The current material durability and texture of the printed model is poor, try to use higher strength and lighter weight carbon fiber composite materials or high strength engineering plastics, to improve the stability of the robot structure and reduce the overall weight, which is conducive to long-term use and mobile scenes.

(ii) Improve the appearance design

- Due to budgetary constraints, we give up the printing of silicone skin, but we can strive for more funding support in the future. Printing silicone skin can significantly improve the degree of anthropomorphism, make it more close to the texture of human skin, and enhance the emotional resonance of the user.

(iii) Development board upgrade

- The response time of the current expression is too long, so we can replace the existing development board with a model with stronger computing power, such as Raspberry Pi 4B (8GB RAM), whose powerful processing power can process voice data and control commands more quickly, and shorten the processing time of the data on the end of the development board.



Raspberry Pi 4B (8GB RAM)

- Optimize servo control hardware: Upgrade the servo driver board, adopt PCA9685 upgraded driver board, support higher frequency PWM signal output, improve the response speed of servo, and make the expression switching more rapid and smooth.

II. System Architecture Optimization

(i) Simplify the data processing flow

- Add local pre-processing function on the ESP32 side to perform preliminary noise reduction and feature extraction on the collected audio data, reduce the amount of data uploaded to the server, and lower the network transmission pressure and server processing burden.
- Adopt multi-threaded parallel processing technology on the server side, assigning tasks such as speech recognition, large model interaction and TTS synthesis to different threads for simultaneous processing, making full use of the computational resources of the server's multi-core CPUs, and shortening the overall processing time.

(ii) Communication protocol optimization

- Evaluate and select more suitable network protocols for real-time data transmission, such as MQTT protocol to replace part of the TCP connection, MQTT protocol has the characteristics of lightweight and low-latency, which can better meet the needs of real-time interaction of robots and reduce the delay of

data transmission.

(iii) Exception handling mechanism optimization

- To enhance the robustness of the system, future development can focus on a more comprehensive error handling mechanism. It is necessary to ensure that the return values of key functions (such as network communication, file reading and writing, servo control, I2S operations) are strictly checked, and the try-except structure is more widely used in Python code to catch and handle potential exceptions. Meanwhile, add a timeout mechanism for blocking operations (such as serial port reading and network connection) to prevent indefinite waiting. Systematically recording error messages will also be helpful for debugging.

III. Algorithm Optimization

(i) Speech recognition algorithm optimization

- SenseVoice ASR and other speech recognition models are lightweighted, and knowledge distillation, pruning and other techniques are used to reduce model complexity and computation volume and accelerate speech recognition speed while ensuring recognition accuracy.

(ii) Large model interaction optimization

- Some of the commonly used big model functions, such as simple dialog logic and answers to frequently asked questions, are deployed on local servers to reduce the number of interactions with the cloud-based DeepSeek LLM API and reduce the impact of network latency on response time.
- For some common user input scenarios, possible response results are generated and cached on the server side in advance, so that when the corresponding user input is received, the most matching response results are obtained directly from the cache and returned to the user quickly to shorten the processing time of the big model.

IV. Usability and Sustainability Optimization

(i) Configuration and modularization

- The hard-coded parameters in the current code, such as API key, server address, port number, pin definition, Angle data of the servo under different expressions, interface text, and various behavioral thresholds (such as VAD sensitivity, blink interval), All should be extracted to external configuration files (such as JSON, YAML or env files) or stored in non-volatile memory on embedded devices. Doing so not only enables the adjustment of system behavior without modifying the core code, but also simplifies the deployment and parameter update processes in different environments. Meanwhile, organize the related functions into independent modules or classes. For example, the steering gear Angle data of different expressions can be structured and stored instead of hardcoding in each expression function.

(ii) Efficient development and collaboration norms

- To ensure the maintainability of the project and the efficiency of team collaboration, it is necessary to standardize dependency management, build processes and version control: In terms of dependency management, for Python

projects, it is recommended to use requirements.txt or Pipfile to lock the environment. For C++/Arduino projects, the library version should be clearly specified and build configurations such as platformio.ini should be provided, along with detailed compilation instructions. Code collaboration needs to be managed through a Git hosting platform (such as GitHub), the submitted information needs to be semantic, and PR/Issue templates should be configured to standardize the contribution process. In terms of security, external input processing and network communication codes need to be reviewed regularly to strictly prevent sensitive information from being mistakenly submitted to the code base.

If you're interested (and we'd be honored), please visit our project repository at https://github.com/YinyuDream/Intelligent-Voice-Interactive-Robots-with-Facial-Expressions to contribute to our project.

5. Management

5.1. Team Structure and Roles

- **Hardware Group:**

Yize Zhao:

As the project leader, responsible for coordinating and scheduling the project progress, and communicating with other team members to complete the tasks. At the same time, was responsible for the design and assembly of the hardware eye part, enabling it to rotate left and right and perform the blinking action.

Wenyi Guo:

As a member of the hardware team, responsible for the hardware modification design and assembly of the eyebrow section. Using the SolidWorks software, I modified the complex eyebrow structure of the original open-source model to make it more capable of simple movement, and completed the eyebrow arching action. At the same time, together with team member Zhidong Zhang, completed the assembly of the overall hardware framework of the robot.

Zhidong Zhang:

As the person in charge of the mouth part of the hardware group, designed and assembled the robot's mouth part. I also modified the modeling of the mouth part in the original open-source model and designed the teeth, gums and other parts using clay materials, making the robot more human-like. At the same time, together with team member Wenyi Guo, completed the assembly of the overall hardware framework of the robot.

- **Software Group:**

Chenhui Liu:

Write the code for controlling expressions. The movements of multiple servo motors with different degrees of freedom were coordinated and arranged, enabling them to perfectly achieve the expressions we needed. Moreover, the most cost-effective servo motor combination was selected within the limited budget.

Pengyu Du:

Write the code for voice interaction. The core functions of the facial expression robot, including the coordination of facial expressions and voice modules, have been realized. By integrating a built-in large model, the robot can intelligently understand the questions raised by customers and provide the most appropriate responses.

5.2. Project Progress Schedule

WEEK 1	WEEK 2	WEEK 3	WEEK 4	WEEK 5	WEEK 6	WEEK 7	WEEK 8	WEEK 9	WEEK 10
Hardware Group									
Understand the concept of the model	Find open-source models on the Onshape website	Analysis model	Modify and optimize the model	Print the model and assemble the model			Conduct functional testing in conjunction with the software		Prepare for the defense
Software Group									
Understand the algorithms related to voice interaction and servo control		Implementation of voice interaction code		Servo control code implementation		Connect the Arduino board for testing	Conduct functional tests in conjunction with the hardware.		Prepare for the defense

- In the **1st to 4th week**, **the hardware team** mainly analyzed, understood and improved the open-source models.
- In the **5th to 7th weeks**, **the hardware team** was responsible for designing and printing as well as assembling the model.
- In the **1st to 2nd weeks**, **the software team** gained a preliminary understanding and learned about the code section.
- In the **3ed to 4th weeks**, **the software team** completed the writing of the sound module code.
- In the **5th to 6th weeks**, **the software team** finished the writing of the servo control code.
- In the **7th week**, **the software team** connected the computer and the Arduino board for testing.
- In the **8th to 9th week**, **the hardware team and the software team** jointly tested the overall performance of the robot.