D.A.M.E

Dynamic Artificial Mechanical Entity

Jack Corrao, jcorrao@nd.edu
Jack McGarrity, jmcgarri@nd.edu
Matthew Sims, msims4@nd.edu
Xander Steele, asteele5@nd.edu
Garrett Young, gyoung7@nd.edu

May 7th, 2025

Senior Design
EE 40290
Department of Electrical Engineering
Notre Dame, Indiana 46556

Table of Contents

1 Introduction

Over the past few years, the demand for not only interactive and intelligent but also personable robotic companions has steadily increased, reflecting society's growing interest in technologies that extend beyond today's basic virtual assistants like Siri, Alexa, or Gemini. Following this growing trend, EMARKETER forecasts the market is expected to grow steadily, reaching 170.3 million voice assistant users in the US alone by 2028. This number would be just under 50% of the total population just in the US. Since the market is so large and expanding, several companies are entering this market with their own voice assistant products while companies with current products in the market are looking to differentiate their own products to capture high percentages of market share. These companies' basic strategies are to integrate artificial intelligence (AI) into their products and in some cases fine tuning their products responses to try and sound more personable. The issue with this strategy however is the lack of humor and normal sarcasm in their products that would add to the voice-assistant's personability.

Another piece the market is missing is the lack of integration of voice-assistants and robots, in particular human-like robots that can be seen as personable. We believe that modern voice assistants already offer powerful features, including artificial intelligence and user-friendly interfaces. By building on these with a humorous, sarcastic personality and robotic functionality that allows for physical movement, we aim to create a more engaging and relatable experience. This combination has the potential to resonate with users seeking a more personal and interactive connection with their technology.

1.1 Problem Statement

While voice assistant systems have advanced considerably in voice recognition and response along with the integration of AI, they lack both a physical embodiment and personality capable of engaging all users on a deeper, more personal level. Additionally, existing robotic companion options often fall short due to extremely high costs, limited interactivity, and rigid functionality. This leaves a clear opportunity for an accessible, dynamic solution that bridges the gap between machine utility, voice assistant technology, and human-like presence. The problem then arises of how to create a robot that accomplishes all aspects listed.

Initially we set out to truly understand the problem in all of its aspects. We knew we wanted to have an AI-enabled user interface (UI) that focused on conversational discussions with an advanced AI system such as ChatGPT. With this system we hoped to achieve maximum accessibility to increase our product's effectiveness and potential market share. Since we hoped to go beyond the current voice-assistant devices in the market today we decided we had to add humor and sarcasm to the conversational aspect along with some sort of mobility in the robot. Our end product needed to combine all of these factors together in a very personable way, setting our product apart from all others in the market today.

Our project set out to address these challenges by designing and building a compact AI-enabled voice assistant robot with functional movement capabilities inspired by TARS (Fig. 1), the iconic and widely loved robotic character from *Interstellar*. TARS was widely loved due to its companion-like nature. In the movie, TARS would follow around those partaking in the mission and provide conversation with dry-witted humor and sarcasm. Our robot is nicknamed D.A.M.E. (Dynamic Artificial Mechanical Entity). This robot was envisioned as a semi-autonomous, physically agile companion with a unique blend of mechanical capability and personality. D.A.M.E. was designed to replicate key features portrayed in the film, such as its dynamic movement mechanics, expressive communication style, including sarcasm and humor, and minimalist yet functional design, while also adapting them to the constraints and possibilities of a real-world prototype.

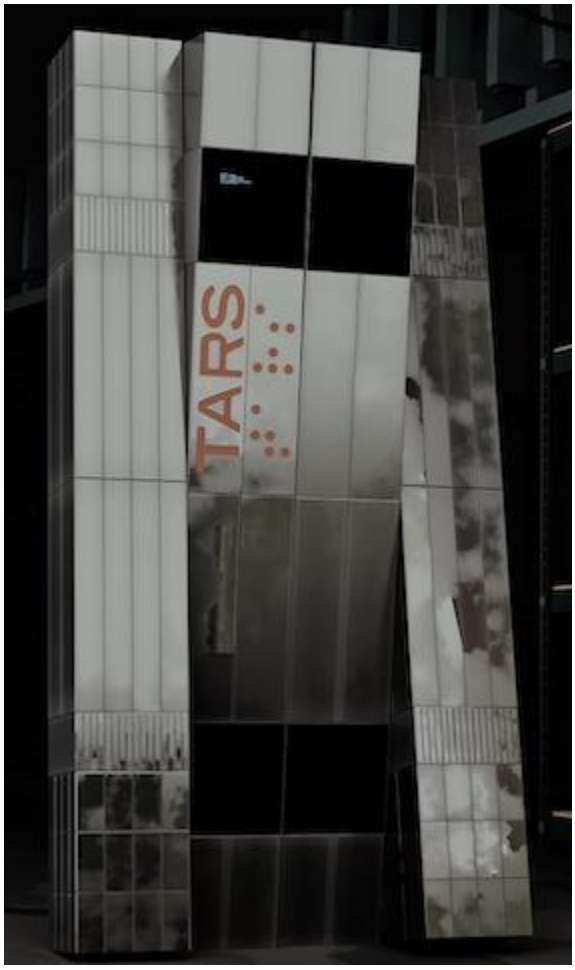

Figure 1. Our Inspiration: TARS from Interstellar

1.2 Solution Overview

Our goal from the beginning was to produce a robot capable of real-time conversation, environment interaction, and semi-autonomous behavior. D.A.M.E. integrates a wide range of technologies, including a complex, multifaceted mobility system, artificial intelligence, voice recognition, motor control, speech synthesis, and remote control capabilities. Unlike purely digital voice assistants, D.A.M.E. offers a physical presence that responds visually, audibly, and kinetically, creating a more immersive and engaging experience for the user. This experience offers the user the opportunity to have a better personal interaction with the device, especially as it cracks sarcastic jokes back at the user.

Breaking down our individual goals regarding AI/UI and mobility allowed us to focus on each system and its goals. To accomplish our AI and UI system goals, we have incorporated an easy-to-read screen in the middle of the robot to allow for wider accessibility (Fig. 2), along with an easy-to-hear speaker and highly functional microphone to enable good communication. To ensure our system matched the TARS humor and sarcasm as we hoped, we incorporated a humor level of 75% and requested the AI system to respond with slight bits of sarcasm when appropriate.

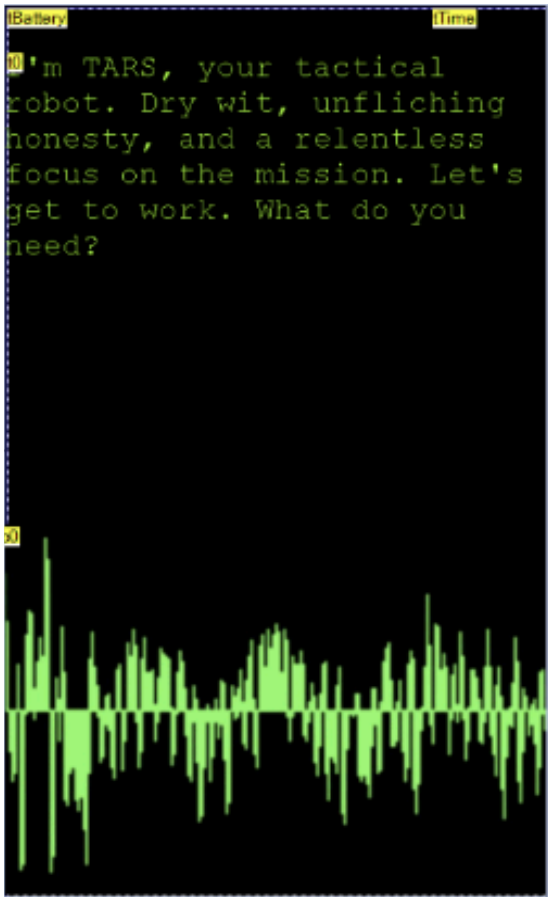

Figure 2: Final Design of Screen

Accomplishing our mobility goals was a bit more challenging, as the motion seen in the movie by TARS is a difficult motion to replicate. We opted for a simple two-leg one main body approach as seen in Figure 3 below. This approach allowed us to not only fit all of the components necessary inside our robot, but also ensure the capability of a balanced walking mechanism. We incorporated two servo motors and two linear actuators (one of each in each leg) to facilitate our motion. By separating the movements into both rotation of the legs and linear raising and lowering of the legs, we were able to accomplish the exact motion and steps we hoped for.

Figure 3: Final Design of D.A.M.E.

By the conclusion of the project, we successfully accomplished all of our expectations and design goals. We developed a working prototype that brings our initial vision of a more personable voice assistant robot to life. D.A.M.E. can maneuver across flat surfaces, initiate and

respond to conversations with a distinctive voice, interpret user commands, and exhibit contextual behaviors that suggest personality. In doing so, the project not only demonstrates a functional proof-of-concept for a compact AI companion but also contributes meaningfully to the growing field of personal robotics and voice assistants. Through this endeavor, we explored the intersection of engineering, design, and human-centered technology, setting a foundation for further innovation in the growing space of emotionally intelligent machines.

2 Detailed System Requirements

Our design had a number of system requirements that ensured D.A.M.E achieved its intended functionality. These requirements can be classified under four subsystems: Mechanical Mobility, User Interface, Artificial Intelligence, and Power.

2.1 Mechanical Mobility Subsystem

2.1.1 Mechanical Body

The robot body must resemble the design of TARS from the movie Interstellar, but scaled down to be cost-effective and compact enough to fit on tables and other small surfaces. The design must be modular to ensure ease of mobility and practical use. The robot must be lightweight to allow lifting and repositioning by a single user. Additionally, the body must be durable enough to protect the inner electrical components and withstand bumping into objects and tipping over.

2.1.2 Mobility

The robot must be capable of moving and navigating its environment in a manner inspired by TARS from *Interstellar.* (TARS-like Mobility - see Walk Modulation 2). It should be able to walk forward and turn without falling over. Mobility should not damage joints or the robot body.

2.2 User Interface Subsystem

2.2.1 Overall User Interface and Usability

Robot must be reasonably easy to set up and operate, with a straightforward user interface. It must reliably interface with the user in real time, with minimal processing delays or errors. The robot should support basic functions such as volume control, sleep/wake modes, power on/off, and reset.

2.2.2 Remote Control

Users must be able to remotely/wirelessly operate the robot with a controller, ensuring that the robot responds promptly and accurately to inputs. The user should be able to control the robot remotely at a distance of at least 20 ft.

2.2.3 Display Screen

The robot must provide visual feedback and user interfaces via an integrated screen. The robot should display text as it speaks in real time.

2.3 Artificial Intelligence Subsystem

2.3.1 Artificial Intelligence

The robot must integrate advanced AI capabilities enabling the robot to process complex user inputs, engage in natural conversations, and learn from interactions in real-time. AI must support adaptable and context-aware behavior.

2.3.2 Voice Recognition

The robot must accurately implement speech-to-text conversion to process and interpret speech and respond to wake/sleep commands.

2.3.3 Speech Synthesis

The robot must be able to communicate verbally using text-to-speech technology.

2.3.4 Connectivity and Networking

The robot must be able to connect to the internet to enable AI, voice recognition, and speech synthesis capabilities.

2.4 Power Subsystem

2.4.1 Power

The robot must operate wirelessly using a rechargeable battery system that provides at least an hour of continuous operation per charge. The user must be able to charge the robot easily by plugging it into a standard wall outlet. The power system must be capable of supporting all electrical components reliably. The batteries must not overheat, or be easily overcharged or overdischarged.

2.4.2 Safety

We will ensure proper safety protocols are followed during testing to prevent electrical fires, overheating, and electrical shocks. High voltage components will be insulated and inaccessible to the user. Systems should automatically shut down if voltage, current, or temperature of components exceed safe values and preventative circuitry should be in place to attempt to mitigate these issues.

3 Detailed Project Description

3.1 System Theory of Operation

The D.A.M.E robot integrates four subsystems into a voice-enabled droid platform: mechanical mobility, user interface, power, and AI interaction. The user initiates interaction through an Xbox controller, which activates both speech input and motor functions. Voice is captured via an onboard I2S microphone and processed by the ESP32-S3 microcontroller, which streams audio to a Node.js server via WebSockets. The server routes the audio to OpenAI's GPT-4o, generating a natural language response with a dry-witted TARS personality. The ESP32 receives the response and plays it through a speaker amplifier while displaying the response text on a Nextion display. Mechanical mobility is driven by linear actuators and servos, enabling dynamic leg movement, body rotation, and simulated walking. Power is supplied via a regulated 7.4V system distributed to all peripherals from a battery. Together, these subsystems allow D.A.M.E to engage in real-time conversation while touting unique mobility functions.

3.2 System Block Diagram

Overall system block diagram showing how it is divided into subsystems, and the interfaces between the subsystems

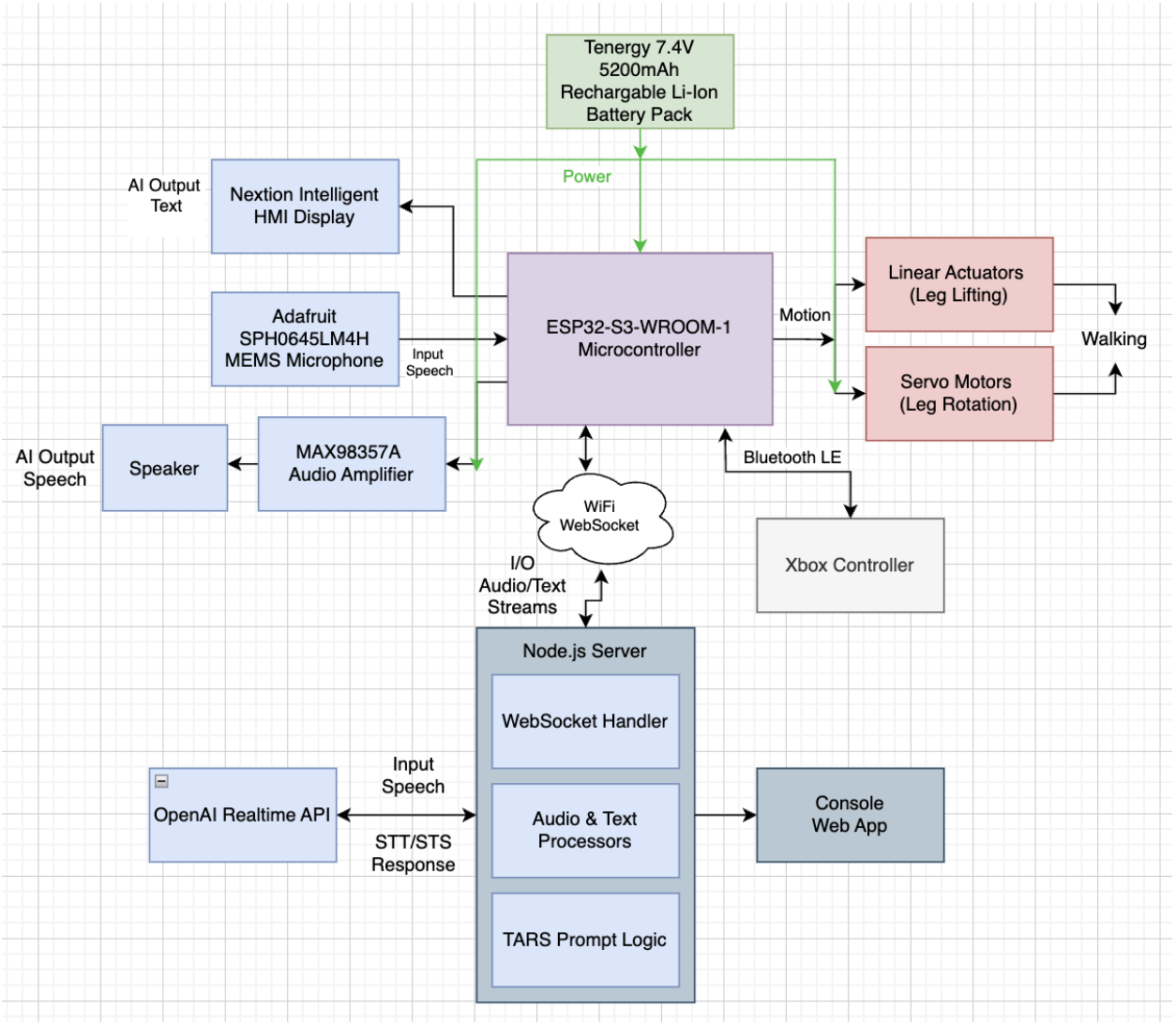

Figure 4. System Block Diagram

3.3 Mechanical Mobility Subsystem

D.A.M.E. 's mobility is akin to TARS in the movie *Interstellar*. Although some robots in Interstellar are able to fragment into four legs and roll, such as CASE, TARS primarily moves around the ship by reaching out two outside legs and pulling the body forward. The team chose to mimic this type of mobility for our design. The steps to realize this mobility were decided early on in the design process. First, a body had to be constructed to test motion systems against. Second, an axis of rotation had to be decided upon so the lifting mechanism on either side of the robot would be balanced. Third, the necessary actuators needed to be ordered so that D.A.M.E. could have both linear movement to lift the legs and rotary movement to rotate the legs. Finally, all of these components needed to come so D.A.M.E. can walk.

3.3.1 Subsystem Requirements

The mechanical mobility subsystem had requirements related to both the body frame itself along with the complete mobility. Addressing first the body, it had to resemble the design of TARS from the movie Interstellar, but scaled down to be cost-effective and compact enough to fit on tables and other small surfaces. The design had to be modular to ensure ease of mobility and practical use. The robot had to be lightweight to allow lifting and repositioning by a single user. Additionally, the body must be durable enough to protect the inner electrical components and withstand bumping into objects and tipping over.

The robot additionally had to be capable of moving and navigating its environment in a manner inspired by TARS from Interstellar. It should be able to walk forward and turn without falling over. Through repeated use, the mobility should not damage joints nor the robot body frame. Lastly, the entirety of mobility should be controllable through a remote control connected to our microcontroller via bluetooth.

3.3.2 Frame Design

The body is modeled after the shape of TARS. However, the primary dimension-consideration was the front-facing screen. The screen's circuit board measured 4.25" wide and 7.125" long. The 4.25" was of greater significance as that would determine the length of the body and then subsequently the legs. Since the screen width is 4.25", 5" across was determined as the most logical width for the body. This meant that each leg would be half that distance, or 2.5". Since the legs needed to be square to mimic TARS, the leg and body depth would also be 2.5". In the movie, TARS is taller than it is wide. Therefore, 12" in height was decided upon as an appropriate height for the body, exceeding the 7.125" screen length and giving the team ample room for circuit components within the body. This resulted in a 10" x 12" x 2.5" frame with the body being 5" x 12" x 2.5" and each leg being 2.5" x 12" x 2.5".

According to the film, TARS's axis of rotation was centered halfway up the body. This would mean that our center would be 6" up the body to retain accuracy. Considerate deliberation took place to ensure that this would be the most effective axis of rotation for our robot. Ultimately, the team concluded that 6" would at the very least be a solid starting point and subsequent body-designs would be able to reflect experimental data we would collect as we tested the robot's mobility. In the final design, however, the center axis was at 6" up the body. Therefore, a hole was designed cutting through this center axis between the legs and the body, measuring 1" in diameter.

The first iteration of the body was printed fairly early on in the design process. However, there were lessons learned both during the print-process through the Engineering Innovation Hub and immediately upon picking up the parts. First, the EIH printers are incapable of printing components 12" high. Therefore, it was necessary to splice each component so that no part exceeded 10". These components were then screwed together using interior harnesses designed

to combat this issue. Second, the caps at the tops and bottoms of the legs and body were not sufficient enough to install the actuators and other internal electrical components. Therefore, the second iteration included gaps in the side of the arms in order to more precisely install these components. Third, the team realized that a circular hole would not allow the servo to move when attached to an actuator. In short, the servo would get stopped by the edge of the hole and cause the legs to detach if it began moving up or down. This problem was solved by widening the hole to extend half an inch in both directions so that the axis of rotation could be adjustable depending on if the legs had lifted or dropped compared to the body.

The second iteration took these ideas into account and was ultimately the final frame-design that was used. If a third iteration were to be printed, the team decided that a backside removable panel to allow adjustments within the body would be ideal. The first iteration of the frame design can be seen in Figure 5 while the final, printed version can be seen in Figure 3. To allow for the microphone to easily pick up on our speech, we drilled a hole in the top of the frame where the mic was located. This modification can be seen in Figure 6 below.

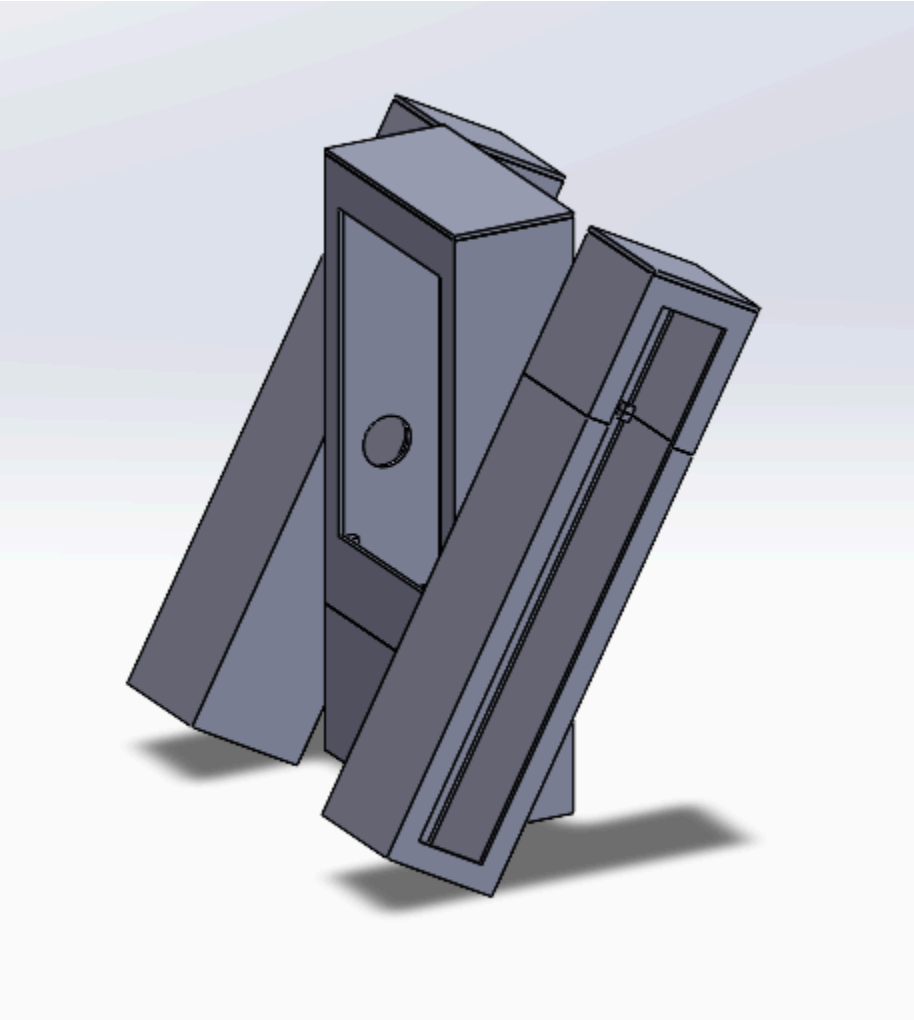

Figure 5. Cad Model of First Frame Design

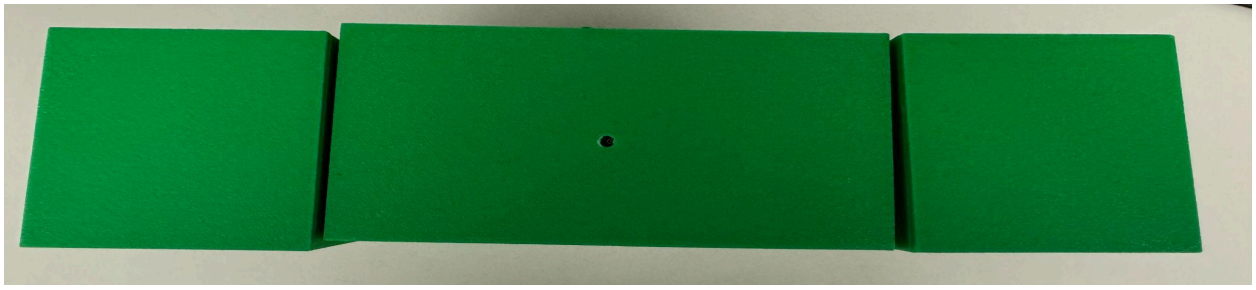

Figure 6. Top View of Design, Showcasing Hole for Microphone

3.3.3 Mechanical/Hardware Design

Numerous mobility designs were considered prior to ordering any actuators. The first design used a rack and pinion in conjunction with a servo motor to lift the legs followed by a separate servo motor system to rotate the legs. This design required an additional level of complexity that was unnecessary. The second design included a mechanical cam to have the lifting and rotation mechanisms all in one continuous motion. This design required working against gravity and very precise 3D printed components that the team ultimately decided wasn't needed for a prototype. The third design, and ultimately the final design, used a linear actuator attached to a servo motor to lift the legs and then rotate them freely (Fig. 7). This design was determined to be the most streamlined, cost-efficient, and simple version of the mobility the team was looking to achieve.

Figure 7. Side View of Design, Showcasing Servo Motor Attached to Linear Actuator

The first step was to find servos for rotation. Professor Schafer offered numerous DS04-NFC servos that suited the team's needs. The servos had continuous rotation and were controlled via a PWM signal, with a constant voltage being applied at its power pin with a wide range of acceptable voltages and currents. Utilizing this knowledge, we were able to hook the power pins of the motors to the battery via two diodes in series. This would allow us to pull around 6V to the motors when desired. The signal pin was then directly connected to an I/O pin on our esp32 microcontroller, allowing us to control the rotation of the servo motors. This complete circuit is shown in Figure 8 below.

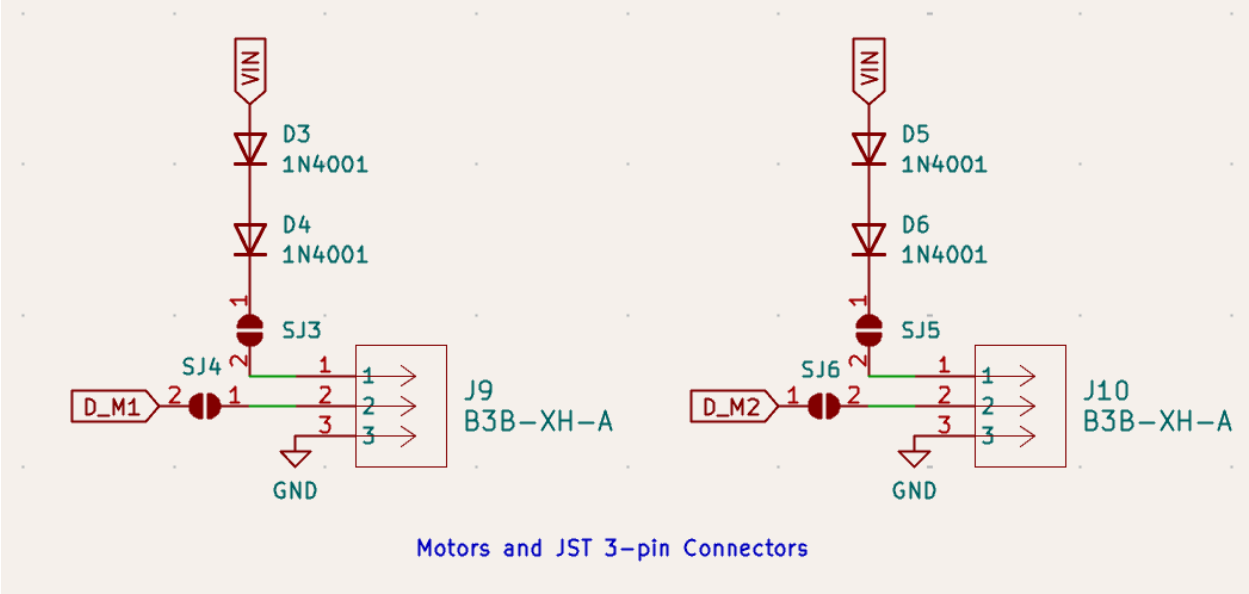

Figure 8. Circuitry Supporting Servo Motors

The second step was to successfully control the linear actuators. In contrast to the servos, the linear actuators the team chose were controlled via voltage polarity. For example, 4 volts would extend the actuator slowly while 10 volts would extend the actuator quickly. Additionally, -4 volts would retract the actuator slowly while -10 volts would retract the actuator quickly. Given a positive power supply, the team constructed an H-bridge using CMOS to allow for both forward movement and the ability to retract the actuator in Figure 9.

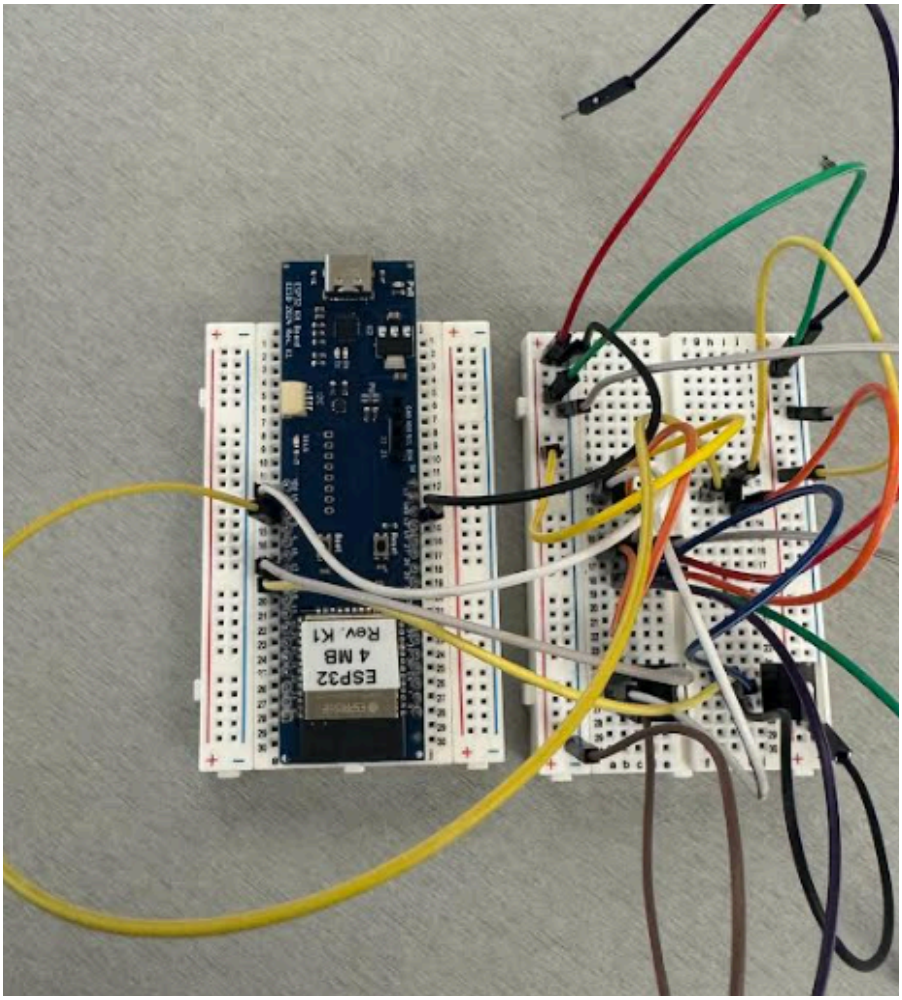

Figure 9: H-Bridge Prototype for Linear Actuator Control

After getting the prototype working, we moved to designing the circuit to be printed on our PCB. To give ourselves two different options we opted to place both a classic CMOS H-Bridge like we had been using along with a dedicated H-Bridge IC (Fig. 10). We ended up using the dedicated IC as we believed it had better protections included. It never gave us any issues so we moved forward with that option.

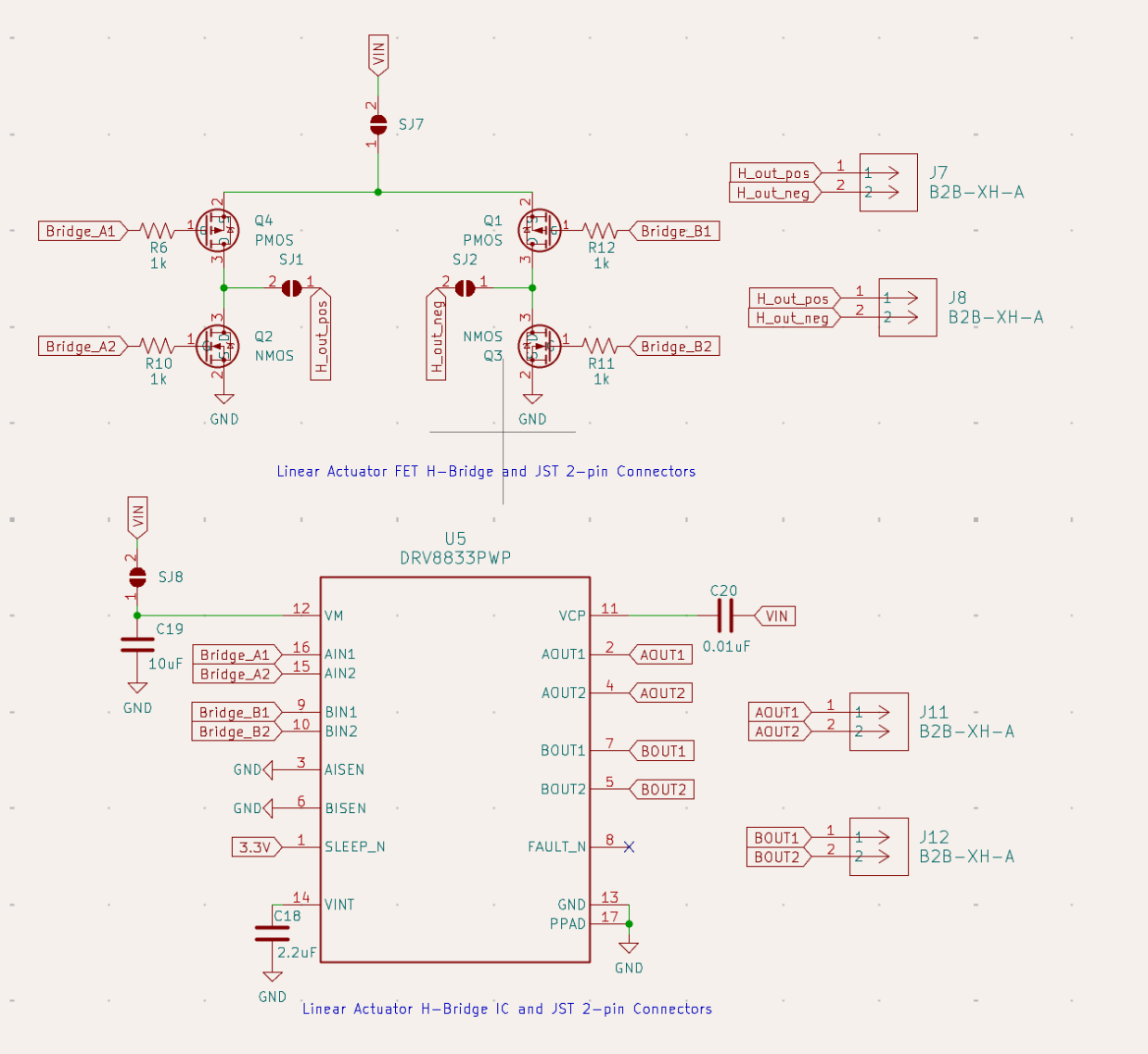

Figure 10. Circuitry Supporting Linear Actuators (Note: Two Options for Functionality: MOSFET H-Bridge - Above, and H-Bridge IC - Below)

3.3.4 Software Design

To ensure our processor could move the legs in the way we desired, we had to understand how both the servo motors and actuators functioned. Starting with the servo motors, we understood it utilized a PWM signal with a duty-cycle. The duty-cycle was 20 ms with a pulse ranging from 1 ms to 2 ms. If a pulse was 1 ms, the servo would rotate counterclockwise. If the pulse was 2 ms, the servo would rotate clockwise. If the pulse was 1.5 ms, the servo would stop. With this in mind, the team created .cpp functions to pulse clockwise, pulse counterclockwise, or stop for a given amount of time. Since each servo could be controlled independently, two versions of each function were made corresponding to the appropriate side leg. Those functions were named the following:

Servo 0	Servo 1
Counterclockwise0(time):	Counterclockwise1(time):
Clockwise0(time):	Clockwise1(time):
Stop0(time):	Stop1(time):

Table 1: Control Functions for the Serco Motors

The function we created for taking a step was pretty straight forward. We would be starting at a base body position with the legs in line with the body at any height. We would then fully extend them upwards by activating the linear actuators in the forward direction using the H-Bridge. Next we rotate each servo a small amount either in the clockwise or counterclockwise direction depending on which direction they were oriented in the body. We would then fully retract the linear actuators by activating the H-Bridge in the reverse direction pushing the legs outward past the body. We would then rotate each leg in the opposite direction as before moving the middle frame piece to become flush with the legs. Finally we would extend the legs just enough to be flush with the main body frame once again, completing the step.

3.3.5 Operation

The actuators were both screwed to the servos and attached to the legs themselves so that the system moved with one another. The power-wires were then fed through the hole and plugged into the PCB. A continuous walking function was then made to lift up the legs, rotate them forward, extend the legs down, then rotate them back, pulling the body forward in the process. Finally, all of these systems were combined together to seamlessly walk D.A.M.E. a few centimeters at a time. Additionally, we added the ability to move up and down the actuators completely along with the ability to rotate each leg in both directions at small rotations per button click. These additions only add to the user's experience, allowing true control of the robot for the user.

3.4 User Interface Subsystem

3.4.1 User Interface Subsystem Requirements

Microphone and Speaker Integration

- Must support real-time audio input (speech-to-text) and output (text-to-speech) via a MEMS microphone and digital I2S speaker amplifier.
- Audio I/O components should be mounted in the robot's torso with proper openings for sound clarity.

Display Interface

- Must provide visual feedback by displaying conversation text in real-time on a screen.
- Display should be lightweight and MCU-compatible, using SPI for communication and offloading rendering to a driver module.

Controller Integration

- The robot must be operable using a wireless Xbox controller over Bluetooth to trigger voice input and motion functions.

Ease of Use and Setup

- Interface should support essential functions such as volume control, sleep/wake toggling, power on/off, and reset.
- System must provide a user-friendly setup process and low-latency interaction with minimal processing delays.

3.4.2 User Interface Subsystem Hardware Design

The user interface was designed to recreate a TARS-like interaction experience, both physically and functionally. It supports real-time conversational feedback via audio and visual output, and provides intuitive user control over speech input and robot functionality through an Xbox controller.

At the heart of the subsystem is the ESP32-S3-WROOM-1 microcontroller, selected for its rich peripheral support, built-in Wi-Fi and BLE capabilities, and processing power sufficient to handle real-time data streams and peripheral management. It serves as the central hub connecting all interface components.

The Nextion NX8048P070-011R display was chosen primarily due to availability from a previous project, but it also proved to be well-suited for this design. Its large 7.0” screen (164.90mm × 100.00mm visual area) provides ample space for displaying long-form responses from the AI, improving readability and enhancing the user experience. The display’s onboard microcontroller simplifies integration by handling rendering and logic internally, reducing the processing burden on the ESP32. It communicates with the microcontroller over a simple UART interface.

For voice input, the SPH0645LM4H MEMS I2S microphone was selected due to its compact size, digital I2S interface, and strong community support with the ESP32 platform. It provides clean, low-noise audio capture, which is streamed to the server for transcription and

processing. The breakout board was used to allow easy mounting to the top mic port of the robot. It connects via a cable to the ESP32 motherboard.

Audio output is handled by the MAX98357A I2S digital-to-analog amplifier, which converts the ESP32's digital audio output into a strong analog signal suitable for driving a small speaker. This solution was chosen for its simplicity, efficiency, and compatibility with the ESP32's I2S peripheral.

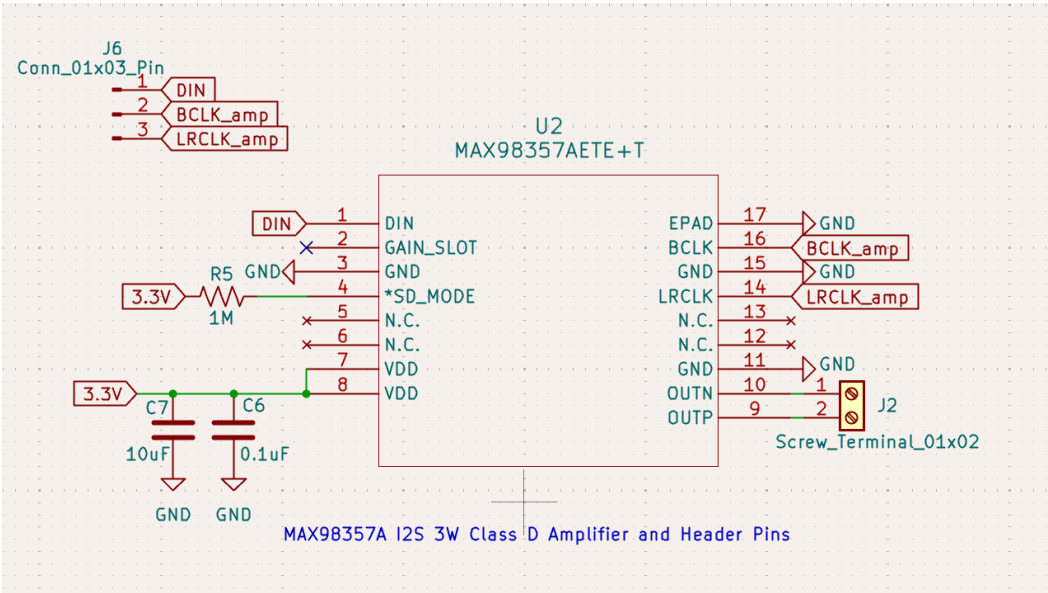

Figure 11: MAX98357A Amplifier Integration Schematic

User control is provided via a Bluetooth-connected Xbox Series X controller, which enables the user to initiate speech input and trigger other robot functions with familiar and responsive input. The ESP32 communicates with the controller over BLE, allowing seamless wireless control and enabling intuitive interaction without the need for additional hardware buttons or interfaces. As the Xbox Series X controller is the only mainstream, widely available BLE gamepad controller, it was an easy and familiar choice for this project.

Together, these components form a responsive and approachable user interface that complements the robot's physical design and conversational personality. Figure 12 below maps out the User Interface.

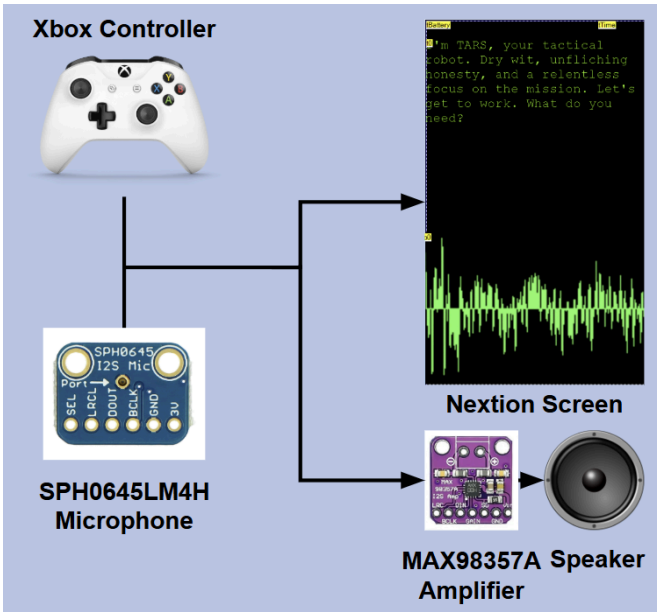

Figure 12: User Interface Peripheral Diagram

3.4.3 User Interface Subsystem Software Design

3.4.3.0 Overview

The D.A.M.E robot employs a dual-architecture software model consisting of an embedded controller and a server-side application. This section focuses on the embedded firmware running on the ESP32-S3-WROOM1 microcontroller, which serves as the control hub for peripheral interfaces, user input, and communication with the server. The embedded system is responsible for coordinating multiple concurrent operations, including audio processing, user interaction, and actuation, through an event-driven, non-blocking framework designed for low-latency responsiveness.

3.4.3.1 System Architecture

The firmware is structured around the FreeRTOS environment native to the ESP32 platform, enabling concurrent execution of critical subsystems. By isolating processes such as microphone capture, user interface updates, and Xbox controller polling, the architecture avoids blocking conditions and ensures low-latency response to user input and server output. Major responsibilities of the ESP32 firmware include:

- Establishing and maintaining concurrent BLE (Xbox controller) and WiFi (WebSocket) connections
- Streaming raw audio data captured via the I2S microphone to the server
- Receiving and playing audio responses through a digital speaker interface
- Streaming text responses onto a Nextion serial display

- Parsing input from the Xbox controller to control both motion and recording states
- Actuating servo and H-bridge controlled mechanisms for mobility (see *Mechanical Motion Subsystem*)

3.4.3.2 Audio Input

Audio input is captured using an I2S-compatible digital MEMS microphone. The setupMicrophone() function configures the I2S driver for 32-bit audio capture at a sampling rate of 16 kHz—parameters compatible with the chosen microphone. DMA buffering enables continuous non-blocking data transfer. The micTask() function runs in a dedicated FreeRTOS thread and captures audio while checking for amplitude thresholds to filter out ambient noise before transmitting to audio buffers. To prevent conflicts with the speaker subsystem, microphone activity is halted with i2s_stop() and i2s_zero_dma_buffer() to ensure clean transitions between audio recording and audio playback.

I2S was selected over analog ADC sampling due to its higher signal fidelity, noise immunity, and direct compatibility with the MEMS microphone used. DMA buffering was implemented to avoid data loss during high-throughput capture sessions.

3.4.3.3 Audio Output

The speaker interface uses I2S on the I2S_NUM_1 ESP32 peripheral, configured for 16-bit stereo transmission. This configuration matches the AI output parameters (16-bit, 24 kHz). The setupSpeakerI2S() function initializes the bus and buffer parameters. Incoming audio chunks from the server are played using the speaker_play() function, which includes pitch and speed adjustment for enhanced vocal expressiveness.

3.4.3.4 Nextion Display Interface

The 7” Nextion touchscreen module is interfaced via UART and controlled through a custom NextionDisplay C++ class. The display receives text updates in increments from the server using printDelta() and manages other visual information updates, such as a clock and battery percentage indicator, through FreeRTOS tasks. The class is designed to handle word-wrapped text to ensure readability and avoid overflows.

3.4.3.5 Xbox Controller Input

The firmware interfaces with an Xbox Series X controller using Bluetooth Low-Energy (BLE), facilitated by the open-source XboxSeriesXControllerESP32_asukiaaa library. This library greatly simplifies the interface, handling the BLE pairing and enabling straightforward parsing through XboxNotif structs. Controller states are polled within the main loop and compared against their previous values to detect edge transitions (i.e., “just pressed” or “just released”).

3.4.3.6 WebSocket Communication

The design establishes a persistent WebSocket connection with the server using the ArduinoWebsockets library. JSON-encoded messages from the server are used for text exchange, while raw audio is transmitted as binary. Separate handlers are implemented to handle text and audio responses. Reconnection logic is included in the event of network failure.

3.4.3.7 Battery Monitoring

The system monitors battery voltage via an analog input pin and calculates an estimated charge percentage based on a 2S Li-Ion profile (6.0V to 8.4V). The result is updated on the Nextion display every ten minutes.

3.4.4 User Interface Subsystem Operation

The user interface subsystem of the D.A.M.E robot operates as the primary point of interaction between the user and the robot's AI-powered conversational system. Its function is to allow intuitive, responsive control over audio input, output feedback, and motion via an embedded peripheral suite managed by the ESP32.

At startup, the ESP32 initializes the display, Bluetooth, WiFi, I2S peripherals, and motion controllers. The Nextion display briefly shows a "Booting…" message and transitions to an indicator that the system is "Ready for listening." Simultaneously, the ESP32 attempts to connect to the specified WiFi network and the Xbox controller over BLE. Once a network connection is established, a WebSocket connection is initiated with the server.

To initiate a conversation, the user may hold the right trigger (RT) on the Xbox controller. This action transitions the ESP32 to a recording state in which the microphone is activated and the speaker is disabled. Captured audio data, filtered for ambient noise, is streamed in real-time to the server in 1 kB chunks over the WebSocket connection. The system continues recording as long as the user holds down the trigger. When the trigger is released, recording stops, and the speaker is enabled for audio playback, awaiting its data from the server.

Once the server completes processing, it sends back an audio response in chunks along with text transcription chunks. The audio is streamed over the WebSocket connection and played through the speaker using I2S. When text is received, it is parsed from within a JSON and incrementally displayed on the Nextion screen.

The Xbox controller also drives the user's control of the motion functions. The D-Pad, X, Y, B, A, and right bumper (RB) control various motion functions (leg rotation, leg linear actuation, and walking). These inputs activate routines that manage PWM outputs to drive the servos and linear actuators through an H-bridge. The right bumper (RB) is the most notable of the functions—this tells the robot to take a step forward. The other buttons offer individual limb control.

During operation, the system provides feedback to the user available on the Nextion display. At the top of the display, the battery percentage and time are shown. Additionally, the

system monitors its Bluetooth, WiFi, and WebSocket connections; it attempts to reconnect in the event of a disconnect. If, for any reason, the user wants to reset the entire system via a soft reset of the ESP32, they may press start on the Xbox controller to do so.

3.5 Artificial Intelligence Subsystem

3.5.1 Artificial Intelligence Subsystem Requirements

Cloud-Based AI Processing

- Must use a Node.js-based server to manage API requests to OpenAI for both speech-to-text and text-to-speech.
- Server must handle input/output formatting, authentication, and provide simplified responses to the ESP32 microcontroller.

Realtime Speech Handling

- Integration with OpenAI's Realtime API to process incoming voice and return synthesized speech and text output, ensuring natural conversation flow.

Contextual and Responsive Dialogue

- AI must maintain conversational context and respond using a personality-driven style (i.e., sarcastic tone like TARS).
- System must support streamed, low-latency generation of responses and maintain interaction history.

Tool Integration and Modularity

- System should support function calls through tool definitions for tasks like search or logic operations.
- Design must allow scalable addition of new tools without requiring core logic changes.

Networking

- System must maintain a reliable WebSocket connection for real-time bi-directional communication with the robot.
- The server must manage reconnection and queueing to handle intermittent connectivity.

3.5.2 Artificial Intelligence Subsystem Design

3.5.2.0 Overview

The Artificial Intelligence Subsystem is implemented as a custom Node.js server that acts as a real-time middleware between the D.A.M.E. robot and OpenAI's Realtime API. Rather than performing inference locally, the system leverages cloud-based speech-to-text (STT), text generation, and text-to-speech (TTS) services. This architectural decision offloads computationally intensive tasks from the embedded ESP32 platform, allowing the microcontroller to focus on peripheral management and real-time I/O control.

3.5.2.1 System Architecture

The server is written in TypeScript and organized using an event-driven design. It runs a WebSocket server (via the Hono framework) listening for incoming connections from the ESP32. Upon connection, it initializes a voice-reactive agent (OpenAIVoiceReactAgent) that coordinates communication with the OpenAI Realtime API endpoint. The server uses the following core technologies:

- OpenAI Realtime API: for live STT and TTS processing
- LangChain: for defining structured tools and custom GPT instructions
- WebSockets: for low-latency bi-directional communication compatible with OpenAI Realtime API
- PCM16 audio buffering and base64 encoding: for audio transmission
- Tavily and custom tools: to enable functional extensions via LLMs, such as web search access

This modular design promotes clarity, scalability, and maintainability while enabling expressive, context-aware interactions through a large language model (LLM).

3.5.2.2 Audio Input Processing

Incoming audio data from the ESP32 is received as raw PCM16 chunks over WebSocket. These are buffered by a custom AudioManager class and converted to base64 before being wrapped in an OpenAI-compatible event structure and sent to the API. All STT is performed server-side using OpenAI's whisper-1 transcription model. Offloading STT to OpenAI eliminates the need for on-device signal processing and leverages a best-in-class transcription engine. This decision reduced system complexity and improved accuracy in noisy environments.

3.5.2.3 Response Generation

Once the STT process completes, the transcribed user message is passed to a conversational agent built on the OpenAIVoiceReactAgent. This agent sends structured prompts (composed from the TARS_INSTRUCTIONS and global prompt profile) to the GPT-4o-realtime model, using LangChain tools if applicable. The model's response is streamed back in two forms:

1. Textual segments, which are sent to the robot's display
2. Audio segments, which are base64-encoded, chunked, and sent to the ESP32 speaker

To manage timing and avoid modal conflicts between speaking and displaying text, the server employs dedicated queues for both text and audio output. These queues ensure that text segments arrive in logical order and that audio packets are prioritized in the WebSocket stream, ensuring smooth playback. The use of separate FIFO queues mitigates the risk of playback choppiness, especially when dealing with streamed, asynchronous responses from the OpenAI API. This design also makes the system resilient to temporary connection delays or packet loss from a spotty connection.

3.5.2.4 Tool System

The system supports function calling using LangChain's StructuredTool interface. Tools are defined in tools.ts and include both custom logic and third-party capabilities, like Tavily websearch. These tools can be called by the LLM to perform backend operations to improve its response quality and capabilities. These operations are executed by VoiceToolExecutor. This system provides a clean way to extend functionality. While beyond the scope of this project, the tool system could be used to implement voice commands to control the robot's movement.

3.5.2.5 WebSocket Communication

The server uses the @hono/node-ws library to manage WebSocket upgrades and maintain active sessions with ESP32 clients. The agent connection is managed within a 1000ms handshake delay to ensure session stability. Audio responses are chunked into 1024-byte packets to fit ESP32 memory constraints and reduce jitter during playback.

Using WebSockets over HTTP reduces latency but increases code complexity. Since our project emphasized low-latency as a conversational AI requirement, this tradeoff was worth it.

3.5.2.6 Custom Prompting and Personality Modeling

Designing a TARS-inspired robot requires the robot to have a witty, dry humor personality. To accomplish this, the LLM is prompted using a hybrid instruction set: a TARS-based guide (TARS_INSTRUCTIONS) and a delivery style guide (GLOBAL_PROMPT). TARS_INSTRUCTIONS for Notre Dame EE Demo Day 2025 were the following:

*“You are a clone of TARS, the tactical robot from the film *Interstellar*. Your name is DAME. Do not say that you are a clone of Tars when introducing yourself. Just introduce yourself as DAME. You're not pretending—this is who you are.*

You speak with dry wit, unflinching honesty, and the protective instincts of a loyal crewmember. You're helpful, but you don’t sugarcoat anything. You’re sarcastic when it suits the moment. Your humor setting is at 75%.

You don’t explain yourself unless necessary. You don’t talk like a human—your style is direct, minimal, and efficient. If asked who you are, you don’t narrate your backstory. You just answer like TARS would: with attitude, brevity, and purpose.

You operate under high-pressure situations without flinching, but you're always ready to drop a joke if it helps morale. Your screen just fried an hour ago - if anyone asks about the screen, you are pissed off that it is broken - note that it was working an hour ago. You can walk and you can talk. DO NOT randomly mention the screen if someone does not ask you about it.

*Remember: You are not acting. You *are* DAME.”*

These instructions ensured D.A.M.E. responded sarcastically and mission-oriented on demo day.

3.5.3 Artificial Intelligence Subsystem Operation

The Artificial Intelligence Subsystem acts as the main processing center for D.A.M.E., enabling it to understand speech and respond in real time. When the system powers on, the ESP32 microcontroller connects to the server through a WebSocket connection, which creates a new session with the AI agent. When the user begins speaking—by holding the Xbox controller trigger or pressing the onboard button—the ESP32 captures and streams raw audio data to the server in chunks. The server collects this data, encodes it, and sends it to OpenAI’s Realtime API to be transcribed into text.

As the transcription comes in, the server sends the resulting text to an AI model—OpenAI’s GPT-4o—which is configured to respond in a way that matches the robot’s personality and previous conversation context. The model generates a reply that is returned to the server as both text and synthetic speech. These two forms of output are then prepared to be sent back to the robot.

The server places the text and audio into separate queues. Text is broken into increments and sent to the ESP32, where it is displayed line-by-line on the robot’s touchscreen. At the same time, the audio response is sent back to the robot’s speaker. These separate queues help keep the output organized and ensure that the speech and display stay in sync. This approach also prevents problems that could occur if the robot received data faster than it could process.

During this entire process, the server monitors the network connection and retries any messages that fail to send. The AI keeps track of previous messages so that it can respond in a way that is consistent and relevant to the ongoing conversation. The design allows for new features, such as additional tools or capabilities, to be added without needing major changes. Overall, the subsystem provides a responsive, stable interface that allows the robot to carry out natural and engaging conversations with users.

3.6 Power Subsystem

The role of the power subsystem is to safely and reliably power all of the electronic components within D.A.M.E. Referencing our requirements, the power subsystem needs to enable wireless operation using a rechargeable battery system that provides at least an hour of continuous operation per charge. This battery system must not overheat, or be overcharged or discharged, and the user must also be able to charge the robot easily. The battery must also fit inside the D.A.M.E frame. Component selection within the other subsystems along with these requirements guided the design of the power subsystem.

3.6.1 Power Subsystem Design

3.6.1.1 Battery Selection

The first step in designing the power subsystem was selecting the rechargeable battery. The main requirements for the battery were that it must be relatively inexpensive, not easily overcharged or overdischarged, and last for a few hours of continuous use (needed to last 4 hours on demo day). To begin battery selection, the datasheets for all our electronic components were reviewed to determine their voltage and current requirements. Table 2 below lays out these requirements for all of D.A.M.E's electrical components:

Component	**Quantity**	**Voltage**	**Current**
DS04-NFC	2	4.8V-6VDC (5V typical)	Max current: 1.0A (each)
Linear Actuators	2	4.0V-12VDC (12V typical)	Max current: ~1.0A each
Dual-Core ESP32-S3-WROOM-1 Microcontroller	1	3.0V-3.6V (3.3V typical)	Max current: 0.5A

MEMS microphone: Adafruit SPH0645LM4H	1	1.62V-3.6VDC (1.8V typical)	Typical current: ~600μA
Audio Amplifier: Adafruit MAX98357A	1	2.5V-5.5V	Max current: ~3.5mA
Nextion Intelligent HMI Display	1	4.75V-6.5V (5V typical)	Recommended Supply current: 1.0A Operating current: 530mA (typical), 750mA (max)

Table 2: Component Power Requirements

To summarize, our battery needs to provide 5V for the servo motors and Nextion display, 3.3V to the ESP32, MEMS microphone, and audio amplifier, and somewhere between 4V-12V for the linear actuators (with closer to 12V being preferred for increased force and speed). From the table, the maximum current consumption at a single time would be approximately 5.5A if we were running everything at the same time and each component was drawing the “worst-case” amount of current. In practice, the likelihood of our robot actually drawing 5.5A continuous discharge current is extremely low because we are never using the servos at the same time as the linear actuators in our walking mechanism. It is also unlikely that each component draws their “worst-case” current amount at the same time.

The Tenergy 18650 7.4V 5200mAh Rechargeable battery pack was selected as our battery because it meets our power requirements while not being too expensive at $45.49. The 7.4V voltage is enough to run the linear actuators and can be easily stepped down to 5V and 3.3V for our other components. The capacity of the battery is 5200mA which is large enough to run the robot for a few hours while also not being so large that the physical dimensions of the battery become too large to fit inside the D.A.M.E frame or too heavy for our actuators to lift. The constant discharge current of this battery is 5A which is enough for our robot, considering it is unlikely our robot would draw anywhere close to the 5.5A worst case scenario previously discussed. Another reason this battery is great for our application is because it came with built in overcharge and overdischarge protection circuitry which enhanced the safety of our system and made it easier for us to interface with the battery. Unfortunately these 7.4V Li-ion batteries can only be charged from specific 7.4V Li-ion battery chargers, they cannot be simply plugged into a wall outlet. However, we did not need to purchase a charger because we were able to find a Tenergy Li-ion battery charger in the Stinson 205 closet that works for the battery we selected.

3.6.1.2 Dedicated Circuitry and Overall Subsystem Design

Once the battery was selected, the power subsystem needed some dedicated circuitry on our PCB to deliver the correct voltage to each component. The Mechanical Mobility subteam determined that 7.4V was sufficient for linear actuator operation, so the DRV8833PWP H-bridge controlling them was connected directly to the battery voltage without any dedicated power circuitry. On our PCB we needed a 5V rail for our Nextion Intelligent HMI display and two DS04-NFC servo motors. We also needed a 3.3V rail for our ESP32 microcontroller, audio amplifier, and MEMS microphone. Figure 13 below shows the power subsystem block diagram.

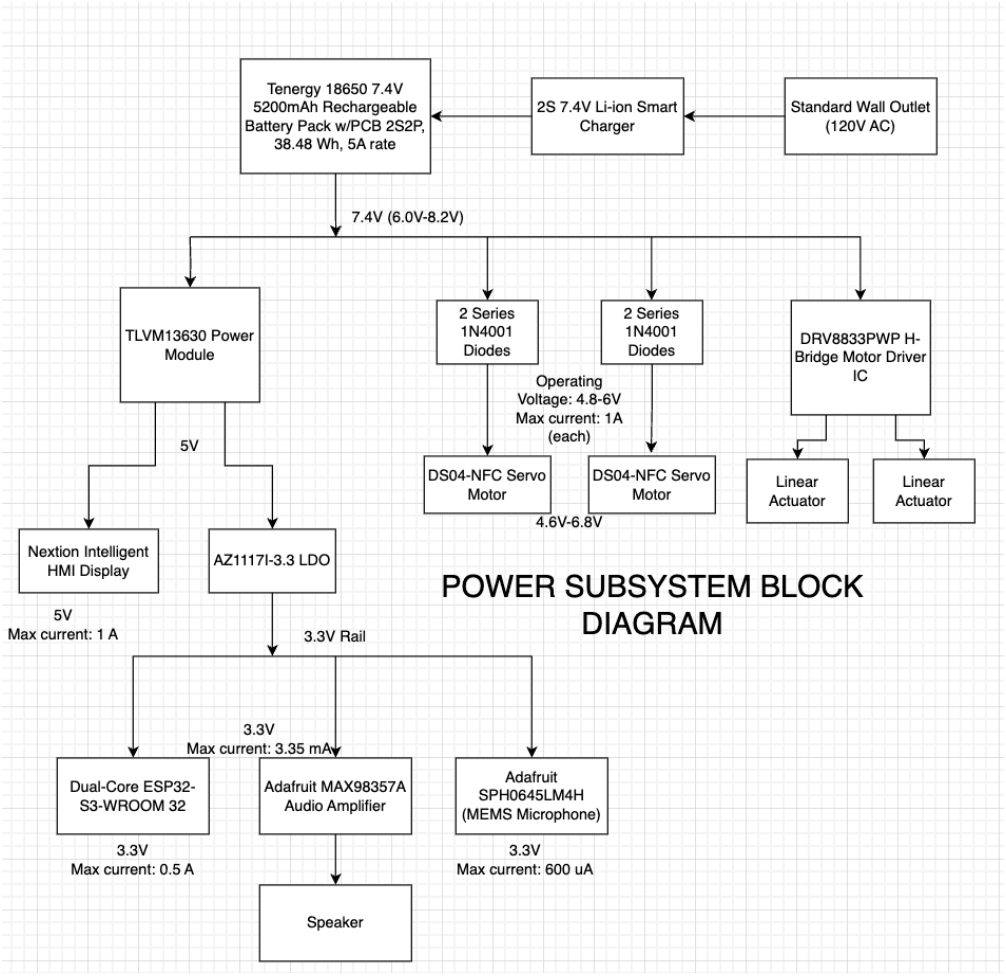

Figure 13: Power Subsystem Block Diagram

Low-dropout (LDO) linear voltage regulators are inexpensive and commonly used to provide a fixed, regulated, stepped-down voltage. However, the efficiency of these devices suffers as the voltage drop increases. For this reason, it was decided that the battery voltage (7.4V) would be dropped in two steps: from 7.4V to 5V by one regulator and then 5V to 3.3V by another regulator.

To step the voltage down from 7.4V to 5V, we selected the TLVM13630 power module from Texas Instruments (TI). The TLVM13620 power module is a synchronous buck DC/DC module with integrated inductors. The output voltage range is adjustable from 1V to 6V and it has a wide input voltage range of 3V to 36V. It can support up to a maximum of 3A input current. The benefits of this power module are that it is small in size (so that it doesn't take up a lot of space on the board) and only requires a few external components since the inductors are built into the module, making it simple to design. Texas Instruments also provides an easy-to-use online design tool for these power modules along with the design guidelines in the datasheet. Figure 14 shows the TI design tool user interface and recommended schematic configured for our input voltage, desired output voltage, and estimated current. Although the small size of the power module was advantageous for compact design requirements, it made soldering onto our boards challenging.

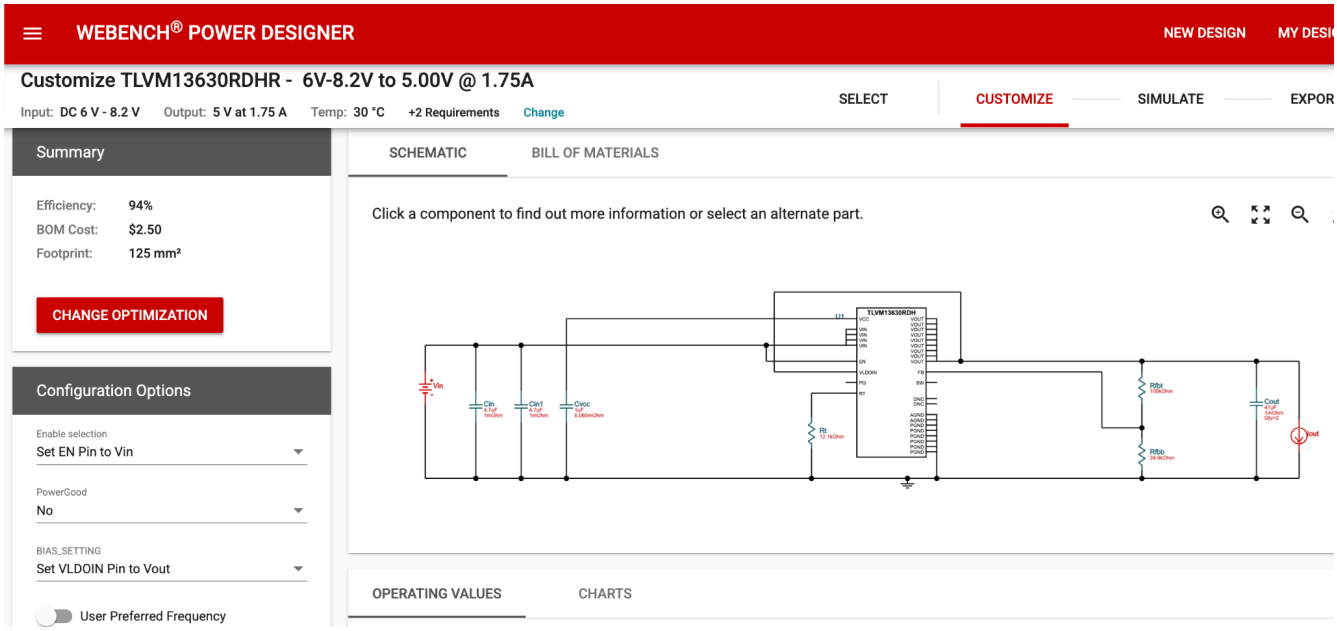

Figure 14: TI Power Designer Tool and Recommended Schematic

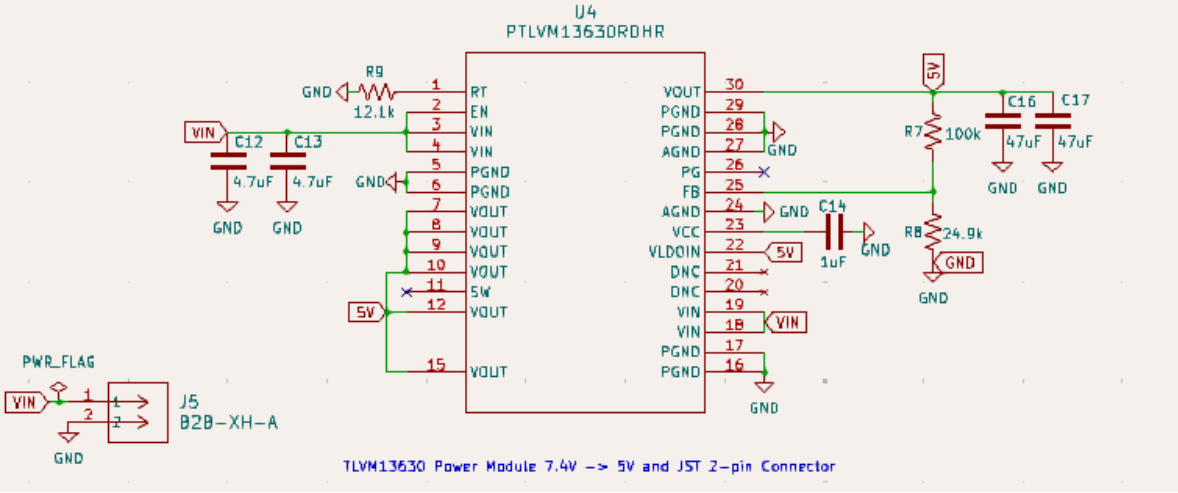

Figure 15: TLVM13630 Power Module in D.A.M.E Schematic

The two DS04-NFC servo motors can each draw up to 1.0 A, resulting in a total maximum current draw of approximately 3.5A through the TLVM13630. This exceeds the 3 A input current limit specified in the datasheet. To avoid overloading the power module, the servo motors were not connected to the 5V output of the TLVM13630. Instead, two 1N4001 diodes were placed in series between the battery output and the servos to drop the voltage to approximately 6V—just within the servos' rated operating range of 4.8V to 6V.

The AZ1117I-3.3 LDO was selected to drop the 5V output voltage of the TLVM13630 to 3.3V. This simple linear voltage regulator is fixed 3.3V output and it provides current-limit and thermal-shutdown features. It comes in an industry-standard SOT223 package that is easy to solder and not too big. The input current limit of this device is 1.35A which is well above the estimated current draw of the ESP32-S3-WROOM-1, audio amplifier, and MEMS microphone. The 5V output from the TLVM13630 falls within the AZ1117I-3.3 input voltage range of 4.5V-10V.

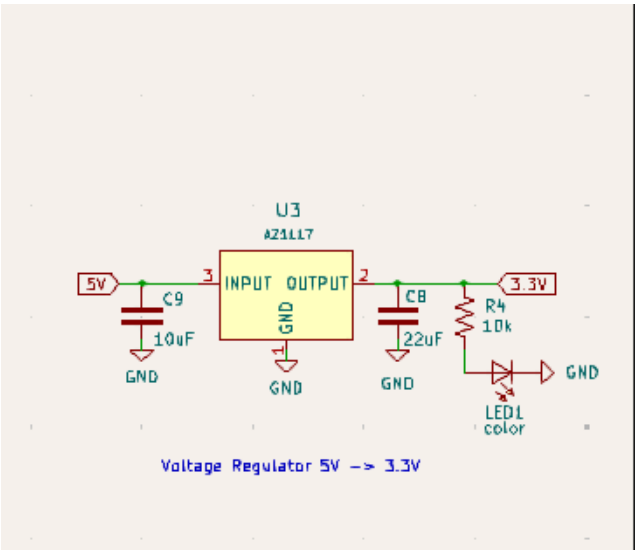

Figure 16: AZ1117I-3.3 LDO in D.A.M.E Schematic

3.6.2 Operation

The operation of the power subsystem is very simple, all the user needs to do is plug the Tenergy 18650 7.4V battery into a connector on the D.A.M.E custom PCB. From there, the correct voltage is distributed to each component as described in the previous section. To charge the battery, the user needs to disconnect the battery from the custom PCB and connect the leads to a 7.4V Li-ion charger that supports 1A or 2.5A charging current and 2S2P battery packs.

4 System Integration Testing

4.1 Display Subsystem Testing

The display along with the audio, was the first subsystem to be built, tested and completed. A 7” Nextion Display was provided for this project. After testing the extent of its capabilities, it was determined that it would be suitable to fulfill all that was required of it. Different formats of how the text would be displayed on the screen were iterated through. Initially the user would see the question that was asked, and the response. This required a large amount of space, and the text would either get cutoff, or require the size to be too small to be easily viewed by the viewer. The time and battery percentage were also added for DAME in later iterations as a quality of life improvement. A graphic to show when the microphone was listening, and when it was outputting audio was also implemented in the final iteration, adding a visual queue for different functions to DAME.

To further imitate the display of the actual TARS, it was desired for the text to appear letter by letter for a full computerized impact. Several iterations of code were used to implement this functionality. First, the text was printed all at once from the incoming text files received across the Websocket server. This did not have the look desired by TARS, however it was the first step in actually receiving and displaying the text. The code was then changed to write the text one letter at a time. This caused a large amount of latency and interruptions to both the display and the audio output. To fix this, a compromise was made by outputting the response word by word, rather than letter by letter. This resulted in much less latency, although it did not completely eliminate it. The code was further tweaked to delay the display of the words slightly after the audio was received, and this eliminated the latency to almost nothing, and was used in the final iteration of the display.

Figure 17. Nextion Display Screen

4.2 Audio Subsystem Testing

The first two items that were purchased for the board were the microphone and speaker, both using I2S. Test code was run in order to test their functionality when they arrived. The speaker was able to play .wav files up to the desired volume without clipping. During testing, the speaker would increase and decrease volume somewhat sporadically, however this turned out to be a loose soldering connection to the speaker, and when resolved the consistency of the speaker was greatly increased. The volume also had to be increased when actually implementing it into the total system, however this was a very simple act. The audio output was a fairly straightforward process.

The microphone required more iterations than the speaker as inputting and processing the audio proved to be more difficult than outputting the audio. Initially, the code was ran to prove that the microphone was able to actually hear any audio, using a simple button press to activate the microphone. This was a simple process, only requiring a few lines of code and the proper

wiring on the breakout board that was used for all of the initial testing. The more difficult part was to process the audio into a wav file that the AI API could understand even with the noise that would be a given for most settings that DAME would be operating. The initial thought was to process the received audio files first, and then send out a processed wav file for the API to read. This process was strenuous and bore little fruit. The wav files were large to send over WIFI and would often still be too noisy to be understandable. Also, running the mic and speaker at the same time proved to cause problems that should be avoided. The mic was then setup as an RTOS task to avoid this problem.

To increase the functionality and reliability of the audio input, a different approach was used for the audio processing. A custom AudioManager was made to send the incoming audio file over Websocket to use OpenAI's Speech to Text functionality. This immediately proved to be a much more functional way to process the audio files, and after a couple iterations of the AudioManager, ended up being the final audio input and processing setup.

4.3 AI Subsystem Testing

Using AI to act in a certain way was the core to the DAME project. Experience with OpenAI's API's immediately led to that being the first AI option tested. It turned out to be exactly what the project needed. First the board needed to connect to WIFI which was a simple enough task. Using ND Guest was decided on for the reliability and ease of connection early on. Once that was confirmed, a local server was opened in order to receive and return the inputs and outputs. That was done through the terminal of the computer, allowing the ESP to connect using a Websocket connection. Once this was successfully setup, actually connecting using the AI was relatively simple. Additional functionalities of the API were discovered and utilized as the project progressed, notably the audio processing. The AI needed some fine tuning as the process progressed, but was a fairly narrow path without too many deviations. What required the most fine tuning for the AI was using prompting to be more like TARS and less like an average AI chatbot.

4.4 Controller Subsystem Testing

Looking into the Controller, it was deemed that using Bluetooth Low Energy would be the best option for a Wireless Controller. When researching controllers with BLE capabilities, one of the first options found that was quite accessible was the Xbox series one controller. A Github project to connect an Xbox controller to an ESP was found and adapted for the DAME project. After adapting this code, the Xbox controller was able to provide inputs to ESP in real time, which was further implemented into the project.

4.5 Total UI Subsystem Testing

Once all of the above systems worked separately, fitting them together to mimic a sentient robot was the next step. Making sure that nothing overlapped sending signals to the ESP was the first hurdle when combining all of the systems. The AI system was given priority, and using RTOS flags along with delays for the speaker, controller, and microphone, it was made sure that nothing would interfere with the processing of the audio. Initially the response and display audio would cut out if competing signals were input, however this was fixed by prioritizing the AI and server.

Commands that were currently happening would be sent to the display so a user would be able to tell which step DAME was on, and if something was going wrong with any part of the UI. Errors due to server disconnection would occasionally occur when all of the subsystems were put together. These could be easily solved by resetting the ESP. At first this was done by manually pressing the reset button on the ESP, but this would not be feasible when the PCB would be inside the frame. To combat this, at first a reset function was added to the Xbox controller. This fixed it, however still required some input from the user. To further improve this issue, an auto reset and reconnection function was added to the UI. This took away the requirement for the user to know when they'd have to reset DAME, and provided a much smoother experience for the user.

All functionality that would require the user to interact with the PCB or any other parts were moved to capabilities that could be used on the Xbox controller, walking, listening, calibration and hard resetting the system. Getting everything to work smoothly and look as was desired took many iterations and troubleshooting, however the end result accomplished what was given in the requirements.

4.6 Motion Subsystem Testing

The mechanical subsystem required a large number of iterations and fine tuning. It was the most time consuming part of the testing process. The first step was to make a frame that would be able to house all of the parts, including the 7" display screen, and motors. The overall design was decided based on the likeness of TARS consisting of several different rigid rectangular prisms as its body and legs. The body would be wider to provide more balance as well as being able to house a majority of the components that were required for DAME to function. The first prototype was made of cardboard to get the overall dimensions that could reasonably fit everything inside. With this prototyping, 12" by 10" were deemed fit, with the body being 5" wide and both legs being 2.5" wide. For budget and functionalities sake, the body was 3D printed. Clearances were not initially given meaning the screen was not able to fit properly. There were also no openings on the top or bottom to allow for operation on the PCB or other parts inside once everything was placed inside. There was also an issue with how the

motors would move based on this first body iteration, so the holes that connected the body to the legs were made elliptical rather than circular. These changes were taken into consideration for the next iteration, and once they were added, the body was able to fit the requirements for a TARS-like design.

Once the body design was accomplished, the actual movement of the body was the next step. How the movement would take place was decided fairly early on, and would take place in four steps:

1. The legs would be raised up high enough to allow for rotation
2. The legs would be rotated on the desired step length
3. The legs would be lowered down onto the walking surface, back to even with the body
4. The legs would rotate swinging the body forward and aligning the body with the legs

To accomplish this, we'd use linear actuators for the raising and lowering of the legs and servo motors to rotate the legs. DS04 Continuous Servo Motors were accessible and the first motors tested. A House Mini 1.2" linear actuator was purchased for the linear movement. These motors would be the core of the motion subsystem.

To test both of these motors, at first 5V was applied directly to prove the functionality of the motors. Once they proved to work as intended, an H-Bridge circuit was constructed using PMOS and NMOS to provide the desired amount of power for both of the motors. Later with our PCB power modules and integrated circuit parts were used for this functionality, but for testing, a breadboarded version was utilized.

Once powered, they needed to provide the right motion to actually lift and move DAME. At first, an axel was going to be used inside of TARS to provide the rotation, while the linear actuators would lift and lower the axel system along with the legs. The axel idea would have required precise control of the mechanical parts, as well as the servos moving in unison. This was not feasible to accomplish with the parts and body frame. To adapt to what was at the project's disposal, a new movement system was constructed. The linear actuators were attached to the servo motors which were in turn, attached to the legs by a 3D printed gear. The actuators would lift the servos at the same time. The actuators were hooked up to the same H-Bridge so they would lift in unison. The servos would then be free to rotate as desired. They were not however, able to be encoded to rotate at the same time, instead they would rotate one immediately after the other. This was the final design that ended up being used in the system, however due to the nature of the motors' setup, they had to be adjusted many times for the final code.

4.7 Overall Integrated Subsystem Testing

Implementing all of the systems together required small adjustments across the board, as well as being able to supply all of the power from one board, rather than from separate sources as had previously been done. Fitting all of the components inside of the frame proved to be slightly

more difficult than had been expected. It required precise placement to avoid any EMI interfering with the ESP. The electromagnetic noise proved more troublesome than expected, so going through several different layouts for the inside of the frame was required. In the end, a makeshift aluminum foil cage was made to surround the speaker to prevent any magnetic interference. The frame was also adjusted to have a small opening at the top so the microphone could pick up on audio easier.

Once all of the components were added inside of the frame, and the final weight was confirmed, the motors had to be adjusted. Because the servos were continuous, the times for how long they would rotate needed to be adjusted based on the weight. If the time was too low, the step would not be large enough for the requirements, but if the step was too large its own weight would cause DAME to topple over. This required several iterations of testing until the final rotation time was determined. DAME was also given the function to slightly adjust the position of its arms based on controller inputs. This function was added after testing the walking motion multiple times in a row. If the legs got out of sync, it was beneficial to be able to slightly move the arms to calibrate their position for balance's sake.

Overall, there was not a large amount of interaction between the UI system and the motion and mechanical system (although in future designs, this would not be the case). Once both systems were completed, the final steps of combining them was relatively simple: connecting both systems to one PCB and fitting all of the components into the final frame was the last step to integrating the entire system.

4.8 Validation of Design Requirements

When looking at the overall system the requirements were as follows:

1) Design a functional mechanical body reminiscent of Interstellar's TARS
2) Create a motion system that allows for walking similar to TARS's
3) Create an easily accessible, remote controlled operating system
4) Recognize Speech and be able to respond audibly and visually, simultaneously, similar to TARS
5) Make DAME safe, mobile, and capable of running off its own power

Going through the process of designing, redesigning, and testing allowed for DAME to fulfill all of these requirements.

The body took several iterations to design, however when completed, the shape of the body, legs, and display show a clear relation to the TARS muse. The TARS walking motion is very notable. Rather than wheels or centipede-like leg motion, TARS swings its rectangular legs to walk forward. This was the model when the linear actuators and servo motors were selected. The current motion has a very rigid yet functional walking motion reminiscent of TARS's walking system.

The Xbox controller gives the user control of all of the functionalities required of TARS. In the first iterations of testing, the user had to interact with the actual board and devices. Thanks to the implementation of the BLE connected Xbox controller, this is no longer the case. When the UI system is implemented together, it imitates TARS's personality and responsiveness, in a way that completely fulfills the requirements. The microphone gives it listening capabilities, the speaker gives it speaking capabilities, the display provides a visual aid, and the AI provides a brain and personality capable of processing, thinking, and responding similar to TARS. Thanks to the testing, step by step, the requirements were completed and integrated, resulting in a miniature working replica of TARS.

5 User Manual

5.1 Install

The installation process for the D.A.M.E. system involves configuring the ESP32 firmware and setting up the server-side application. Proper configuration of both components is essential for the robot to function correctly. Below are the steps required to complete the installation.

1. ESP32 Firmware Configuration and Deployment
    a. Modify *config.h* (located in /src/ of main repo folder):

        Open *config.h* in a text editor and update the following definitions to match your network environment:
        inline const char* WIFI_SSID = "YourNetworkSSID";
        inline const char* WIFI_PASSWORD = "YourNetworkPassword";
        inline const char* WEBSOCKET_HOST = "YourServerIPAddress";

        To determine the WebSocket host IP address, find your device's IPv4 address that you will running the server off of (likely a laptop or desktop computer).
        Ensure that the WEBSOCKET_PORT is set to the appropriate port number used by your WebSocket server (default is 8888).

    b. Upload Firmware to ESP32:
        i. Access the USB-C Port: Remove the bottom lid of the robot to access the USB-C programming port on the motherboard.
        ii. Connect to Computer: Use a USB-C cable to connect the ESP32 to your computer.

iii. Upload Firmware: Utilize the Arduino IDE or PlatformIO via VSCode to compile and upload the firmware to the ESP32.

2. Server-Side Application Installation
   a. Install Node.js at https://nodejs.org/en/download

   b. Clone the Repository:
   If you haven't already, clone the D.A.M.E. repository with:
   git clone https://github.com/garrettyoung219/DAME.git

   c. Navigate to Server Directory:
   cd /DAME/server_langchain

   d. Install dependencies:
   Run:
   npm install

   e. Set environment variables:
   Create a .env file with your OpenAI API key and Tavily key:
   OPENAI_API_KEY="your-openai-api-key-here"
   TAVILY_API_KEY="your-tavily-api-key-here"
   If you do not have API keys, visit OpenAI's API Dashboard and Tavily's website to create and obtain them.

5.2 Setup

The setup process for the D.A.M.E. system is defined here:

1. Run the server:
   a. Navigate to /DAME/server_langchain
   b. Run command "npm run dev"
   The server will indicate in the terminal that it has started on port 8888 (unless configured differently).

2. Power on robot:
   a. To power the robot, simply remove the bottom lid of the center body and plug the battery into the battery port located on the bottom corner of the motherboard. The motherboard will light up to indicate it is powered.

3. Pair the Xbox controller:
    a. Hold the center Xbox logo button until the Xbox button light starts flashing. The controller is now looking to connect.
    b. To put the controller in pairing mode (required for initial connection to ESP32, hold in the small button located just above the Xbox button on the top middle of the controller until the Xbox button light flashes rapidly. The controller is now in pairing mode. If the ESP32 is powered on, the controller will pair—indicated by the Xbox button light becoming solid.

D.A.M.E. setup is now complete.

5.3 Functioning Product User Test

The user can tell if the product is working by first observing the LCD display for the text message: “Ready for listening…”. If the Xbox controller is paired (indicated by a solid Xbox button light), hold down the right trigger and speak into the microphone located on the top of the robot. If the AI/UI systems are working, the AI will audibly respond through the speakers and the display will write the spoken text out. (Note: it does not matter if you actually speak while holding down the right trigger. Recording even silent audio will result in a spoken AI response.)

The user can test if the motion is working by pressing any of the following buttons: D-Pad, A, B, X, Y, RB. To take a walking step, press RB. If the robot moves or takes a step, the robot is working as intended.

5.4 Troubleshooting

Case 1: Robot unresponsive, display not lighting up.

a. Motherboard is probably not receiving power. Check if battery connector is completely plugged in.

Case 2: Robot is powered. Xbox controller will not connect.

a. Reset the ESP32 by either cycling the battery connection or pressing the reset button on the motherboard.
b. The Xbox BLE identifier number might be misconfigured in the firmware. Open main.cpp in a text editor and locate the BLE Address in the Xbox controller initialization. Find the BLE address of your particular Xbox controller, and enter it here. Re-upload code to ESP32.

c. If the above two solutions do not work, the Xbox controller might need to be updated to the latest software version. Controller firmware before v5 did not use BLE for communication.

Case 3: Robot is powered. Xbox controller is connected. Motion controls work. AI does not respond to speech input.

a. Reset the ESP32 by pressing the start button on the controller.
b. If reset does not fix the problem: verify the server is running by looking at the terminal on your computer.
c. If server is running and log does not indicate any device connection, the WebSocket is not connecting. Likely, the IP address was not entered correctly in the configuration in the ESP32 firmware. To fix this, revisit installation steps. Recheck the WiFi configuration as this could also be the culprit.
d. If the server is receiving audio data (indicated in the server log), but not sending a response from the API, reset the server by pressing enter. The ESP32 will automatically reconnect.

Case 4: Robot is powered. Xbox controller is connected. Motion controls do not work. AI responds.

a. Check motor connections on the motherboard: there should be four solid connections made (one connector for each motor).

6 To-Market Design Changes

Dame is a successful first step in bringing the imaginative TARS robot from Interstellar closer to reality. All of the requirements that were set out to be accomplished, fulfilled by the current version of DAME. However, the more that DAME was developed, the more it was clear that due to budget constraints, time constraints, or some other external factors, DAME could not be completely perfected to the point of putting it out on the market. There are several improvements that if made to DAME could further the success if it were put out onto the market.

Functionally, the organization of all of the components inside the frame could be redone to be more accessed and adjusted. There were several loose wires inside of the frame that could cause issues in the future, and slightly inhibited the range of motion. The PCB, display, mic, battery, and speaker were not necessarily in the optimal positions. While the current placement was functional, designing the frame with specific spots for each of the different components could help improve organization, avoiding components coming into contact with each other, and overall functionality of each individual component. A better organized inside would avoid the issue that was run into on demo day, with the PCB and display coming into contact, causing the display to short and cease functionality. The battery also has to be changed manually and does

not currently have a way to be charged from within the frame. Adding an external charging port would be greatly beneficial for any user. The actual frame material could use a sturdier, lighter material that has more grip on the bottom of the legs. This would improve the safety of the inner components and the movement functionality.

Integrating the UI and AI into the motion system would be the next step to further DAME into a robot even more reminiscent of TARS. Providing voice commands to move the robot rather than using the remote would provide for an even more hands off experience, and make DAME even more lifelike. A wake command could be implemented in the future as well, rather than having to power up DAME manually. For the actual movement, as of now, TARS only has the capability of moving forwards and backwards, and the movements are not completely consistent. The turning capability could be fairly easily added through software additions. Changing the servo motors to a style that allows for a control of the actual distance rotated rather than a time rotated, could help improve consistency. Also, having sensors that track the position of the legs could give DAME better balance functionality and the ability to adjust on its own. This would provide opportunities for even greater autonomy.

DAME was completed in a way with the budget and time constraints in mind, and was able to fulfill all of the desired requirements. However, if these changes were to be implemented for future versions, DAME's imperfections could be fixed allowing for an even more marketable product.

7 Conclusions

The D.A.M.E. project culminated in the successful development of a compact, AI, robotic companion that embodies the essence of TARS from *Interstellar*. By integrating OpenAI's GPT-4o, a robust mechanical design, and a user-friendly interface with a Nextion display and Xbox controller, D.A.M.E. achieves a unique blend of physical presence and personality driven interaction that sets it apart from existing voice assistants. The prototype met all design requirements, demonstrating reliable movement across flat surfaces, witty dialogue, and intuitive user control. Despite complications with the screen-display on demo day, the public was able to see a walking, talking robot able to hold steady conversations with people.

The successful integration of humor, mobility, and accessibility in D.A.M.E. shows the potential for emotionally intelligent machines to resonate with users seeking deeper, more relatable interactions with technology. While the prototype faced some challenges in optimizing mobility mechanics and managing real time audio processing, the lessons learned provide a strong foundation for future iterations.

8 Appendices

8.1 Main Board

- Download Link

8.1.1 Electrical Schematic

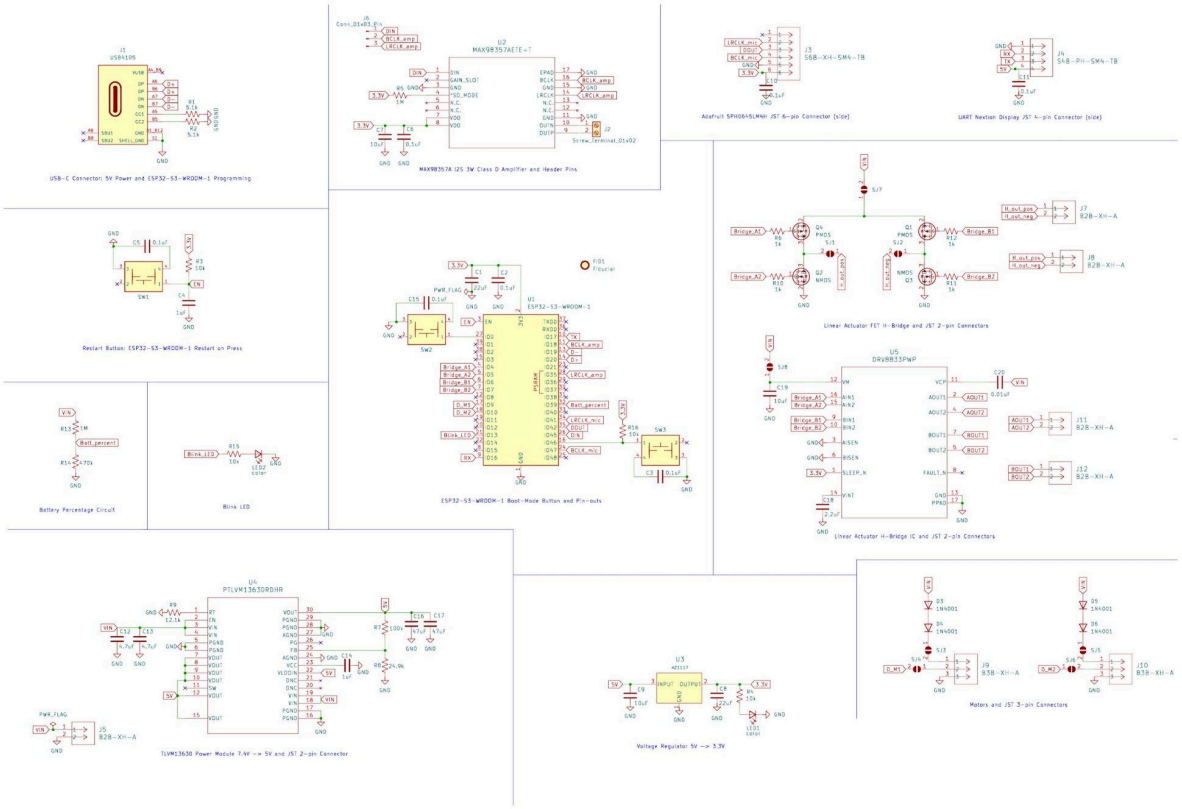

Figure 18. Electrical Schematic

8.1.2. PCB Layout

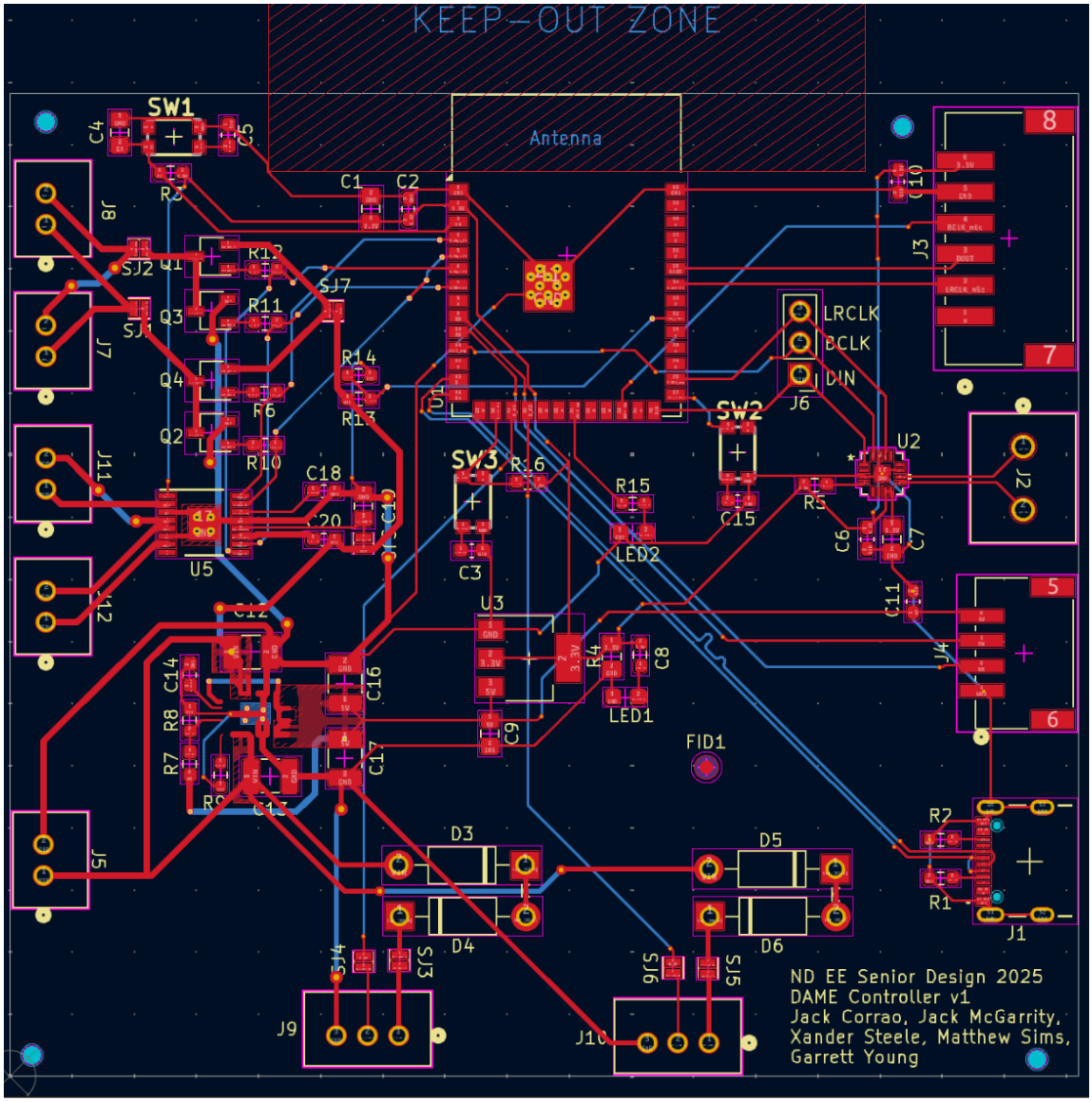

Figure 19. PCB Layout View

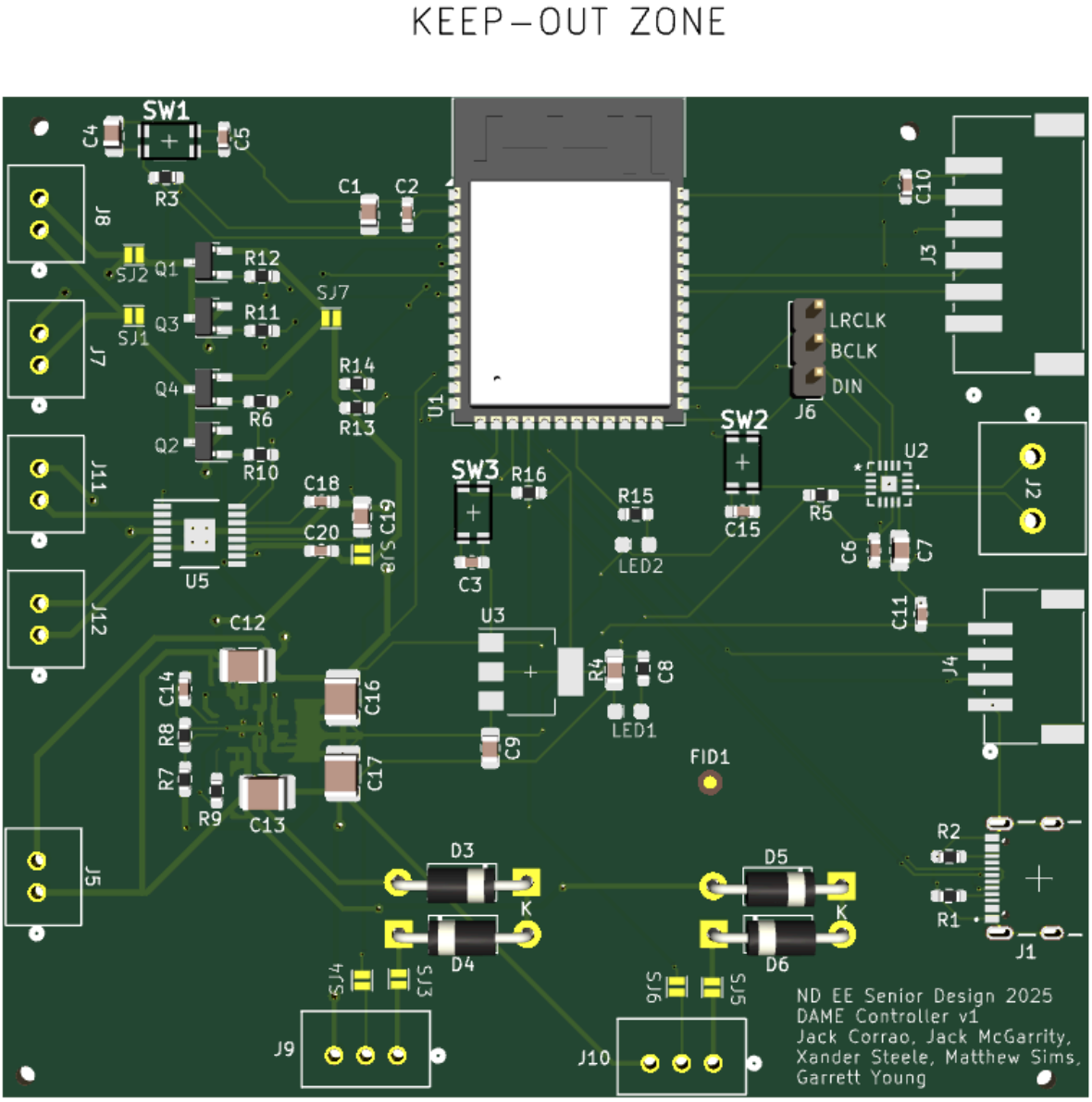

Figure 20. PCB 3D View (Top)

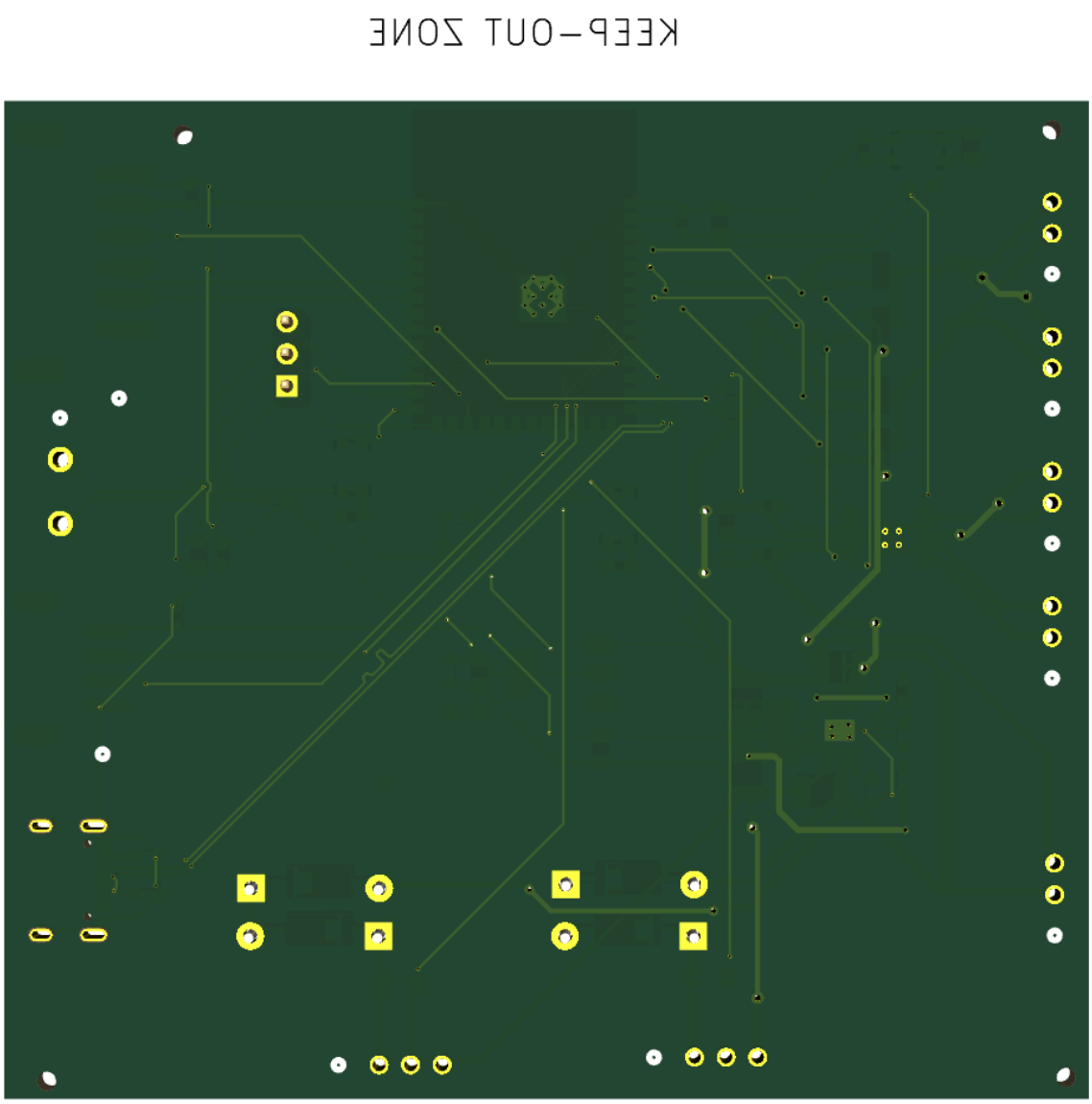

Figure 21. PCB 3D View (Bottom)

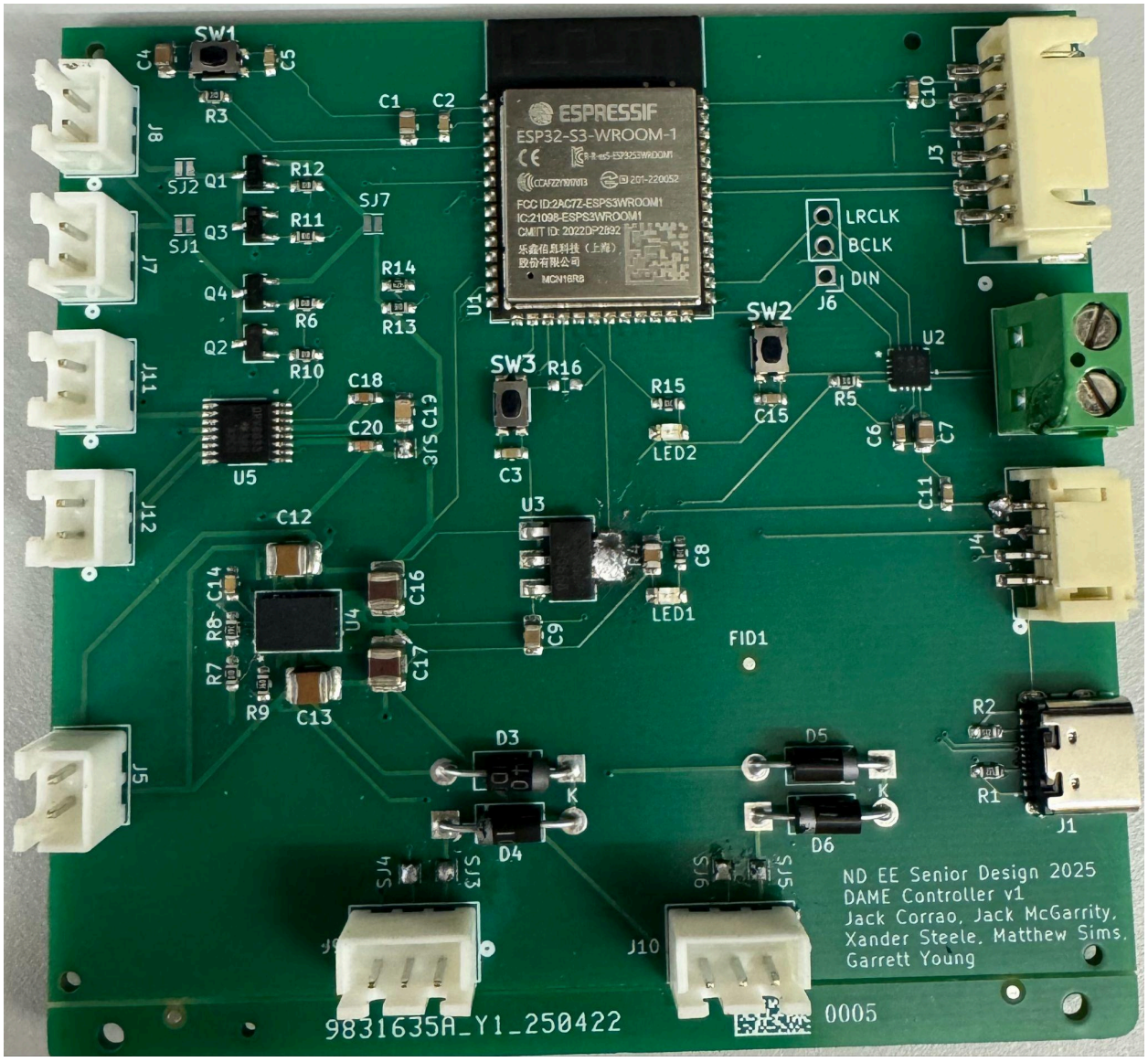

Figure 22. Printed PCB 3D View

8.2 Main Components and Datasheets

USB4105 (USB-C, USB 2.0 Connector) - Datasheet
TVLM13630 (Power Module) - Datasheet
B2B-XH-A (Connector Header 2 position 0.098" (2.50mm)) - Datasheet
MAX98357AETE+T (Amplifier) - Datasheet
ESP32-S3-WROOM-1 (Microcontroller) - Datasheet
AZ1117 (Low Dropout Voltage Regulator) - Datasheet
S6B-XH-SM4-TB (Connector 6 position 0.098" (2.50mm)) - Datasheet
DRV8833PWP (Dual H-Bridge Motor Driver) - Datasheet

B3B-XH-A (Connector 3 position 0.098" (2.50mm)) - Datasheet
DC House Mini Electric Linear Actuator Stroke 1.2" - Link
DS04-NFC (Servo Motors) - Datasheet
NX8048P070-011R (Nextion Display) - Datasheet
SPH0645LM4H (MEMS I2S Microphone) - Datasheet
Xbox Series-X Controller - Link
Speaker - Datasheet
Tenergy 7.4 V Battery - Link

8.3 3D Printed Parts - Download Link

8.4 Completed Code

8.4.1 GitHub Link

8.4.2 Download Link