Voice-Controlled Robotic Arm with Biometric Authentication System

Report submitted to the SASTRA Deemed to be University as the requirement for the course

IDP401M–**MINI PROJECT**

Submitted by

NAME: Arthi Priya.A

(Reg. No. : 126011004 , B.Tech Bioengineering)

NAME: Charookiren G L

(Reg. No. : 126018008 , B.Tech Computer Science and Business Systems)

NAME: Sanjai S

(Reg. No. :126018042 , B.Tech Computer Science and Business Systems)

November 2025

SCHOOL OF MECHANICAL ENGINEERING

THANJAVUR – 613 401

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr.S.Pugazhenthi**, Dean, School of Mechanical Engineering, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan,** Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Student Welfare.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Mechanical Engineering who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from our family and friends resulting in the successful completion of this project.

We thank you all for providing me an opportunity to showcase our skills through project.

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF MECHANICAL ENGINEERING

Declaration

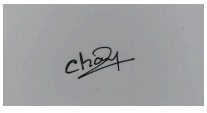
We hereby declare that the project titled "**Voice-Controlled Robotic Arm with Biometric Authentication System**" is an original work carried out by us as part of our academic requirements. The work presented in this report is based on our own effort. Wherever ideas, data, or materials from other sources have been used, appropriate acknowledgement and citations have been provided.

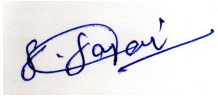
This report has not been submitted, either in part or in full, for the award of any degree, diploma, certificate, or any other academic recognition at any institution.

Declared by

Date: 13.11.2025

Signatures :  **(Arthi Priya A [126011004])**

 **(Charookiren G L [126018008])**

 **(Sanjai S [126018042])**

List of Figures

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Context and Background

Voice-controlled robotic arms have been evolving fantastically from basic to more advanced command-driven models that enable usability, accessibility, and independent living in the real world. The foundational research in this domain, such as the "Development of a Voice Controlled Robotic Arm," established a baseline using adaptive neural networks and custom inverse kinematics on a PIC microcontroller, achieving high accuracy.

Subsequent research has largely converged on increasing accessibility and reducing cost. Many recent works have explored similar directions, creating low-cost, 3D-printed robotic arms controlled by smartphone applications, leveraging the Arduino Nano and servo motors to provide affordable, hands-free control for people with physical disabilities. Other studies have focused on industrial applications, using Arduino and Bluetooth modules to fabricate manipulators for hazardous material handling and remote operations. This trend highlights a consistent research goal: the democratization of robotic technology through low-cost, open-source platforms such as Arduino, Raspberry Pi, and Android mobile apps, which offer adaptability for both educational and assistive applications.

The important thrust has been on user-centered design, with projects illustrating modular architectures that can be tailored for prosthetic applications or complex industrial manipulation. Improved adaptability via machine learning that enables systems to learn voice patterns, robust wireless or mobile integration, and the ability to expand to multimodal control—voice, gesture, vision—are key advantages across these works.

However, as these systems move from academic proofs-of-concept to functional assistants, the limitations of simple command-and-control paradigms become apparent. The next frontier in Human-Robot Interaction (HRI) lies in developing systems that are not merely "voice-operated" but "intelligent." This involves integrating advancements in artificial intelligence, such as large language models (LLMs), to make robots easier to use and more intuitive. Modern HRI research is exploring systems for social assistance, healthcare, and education, where robots must understand complex, natural language commands, operate safely in shared human spaces, and manage interactions with multiple users.

1.2 Problem Statement

The initial development phase of this project, documented as a baseline system, successfully demonstrated a functional voice-controlled robotic arm using an Arduino microcontroller and standard Python libraries. This "Phase 1" system served as a valuable proof-of-concept but also exposed critical limitations that prevent its viable deployment in a real-world, user-centric application. The problem statement for this project is defined by four key deficiencies in the baseline system:

1. **Security Vulnerability:** The baseline system is fundamentally unsecured. It processes commands from any user within microphone range. In any target application—from an assistive device for a user with disabilities to a shared lab instrument—this open-access model is untenable, as it permits unauthorized or malicious control.
2. **Robustness Deficit:** The system relies on a standard Python SpeechRecognition library, which, while functional, exhibits significant performance degradation in non-ideal acoustic environments. The simple ambient noise calibration is insufficient to handle real-world challenges like background conversations, household appliances, or variable microphone distances, leading to a high Word Error Rate (WER) and frequent command failure.
3. **Usability Constraint:** The command-and-control logic is rigid. The Python script is hardcoded to recognize a small set of single-word commands (e.g., "up," "down"). This interaction model is unnatural and brittle. It lacks the capacity to understand natural language or parse complex, parameterized commands (e.g., "move the arm left 45 degrees"), which is a primary requirement for a truly "intelligent" assistant.
4. **Reliability and Safety Gap:** The baseline system operates entirely in an "open-loop" control mode. The Arduino issues a command to the servo (e.g., servo.write(90)) and then *assumes* the action was completed. The system has no "ground truth" awareness of the arm's actual physical position. This blindness means it cannot detect if the arm is physically blocked (a stall), if it has been moved by hand, or if a command would force it beyond its physical limits. This lack of feedback is a critical safety and reliability failure.

1.3 Project Contribution and Objectives

This project directly addresses the problem statement by designing, implementing, and analyzing a multi-layered software and hardware architecture that transforms the baseline prototype into a secure, robust, and intelligent robotic system.

The objective of the baseline project was to design and develop a voice-controlled robotic arm using three servo motors that can move left, right, up, and down based on simple voice commands, processed by a microcontroller.

The primary objective of this *project* is to substantially expand this concept by engineering a holistic HRI framework. The core contributions of this work are the design and integration of five novel subsystems not present in the baseline:

1. **Biometric Speaker Verification:** To solve the security vulnerability by implementing a voiceprint authentication layer, ensuring only pre-authorized users can access the arm's control functions.
2. **Multi-User Arbitration:** To manage the system in a shared environment by developing a "first-come, first-served" session lock protocol that grants exclusive control to a single authenticated user at a time.
3. **Advanced Audio Processing Pipeline:** To enhance robustness by integrating an on-device, "always-on" hotword detector for activation and leveraging a cloud-based API (VAPI) for superior noise suppression and speech-to-text.
4. **LLM-based Task Orchestration:** To overcome the usability constraint by utilizing a large language model (Google Gemini) for Natural Language Understanding (NLU). This enables the system to interpret complex, parameterized, and ambiguous user commands, translating them into structured, actionable data.
5. **Closed-Loop Hardware Control:** To close the reliability and safety gap by modifying the hardware to read the servo's internal potentiometer, providing the Arduino with real-time positional feedback. This "ground truth" data enables state-aware error handling, limit-checking, and stall detection.

1.4 Report Structure

This report details the evolution of the project from a simple prototype to an advanced, Project-level system.

- **Chapter 2** presents the architectures of both the Phase 1 (Baseline) and Phase 2 (Advanced) systems, offering a direct comparison.
- **Chapter 3** provides a deep dive into the security framework, detailing the theory and implementation of voice biometric authentication and the multi-user arbitration logic.
- **Chapter 4** details the advanced audio and command-processing pipeline, from on-device hotword detection and noise suppression to the use of VAPI and Google Gemini for NLU.
- **Chapter 5** focuses on the hardware and microcontroller-level advancements, explaining the implementation of closed-loop positional feedback and the state-aware, error-handling logic.
- **Chapter 6** discusses the results of the baseline system and provides a critical analysis of the projected performance, advantages, and trade-offs of the advanced project system, concluding with future work.
- **Chapter 7 & 8** provide the consolidated references and a comprehensive appendix with code for both the baseline and advanced systems.

CHAPTER 2:Objectives and SYSTEM ARCHITECTURE AND IMPLEMENTATION

This project's development is bifurcated into two distinct phases. Phase 1 represents the initial proof-of-concept (a "Mini Project"), which established basic functionality. Phase 2 represents the advanced system developed for this project, which re-architects the entire framework to be intelligent, secure, and robust.

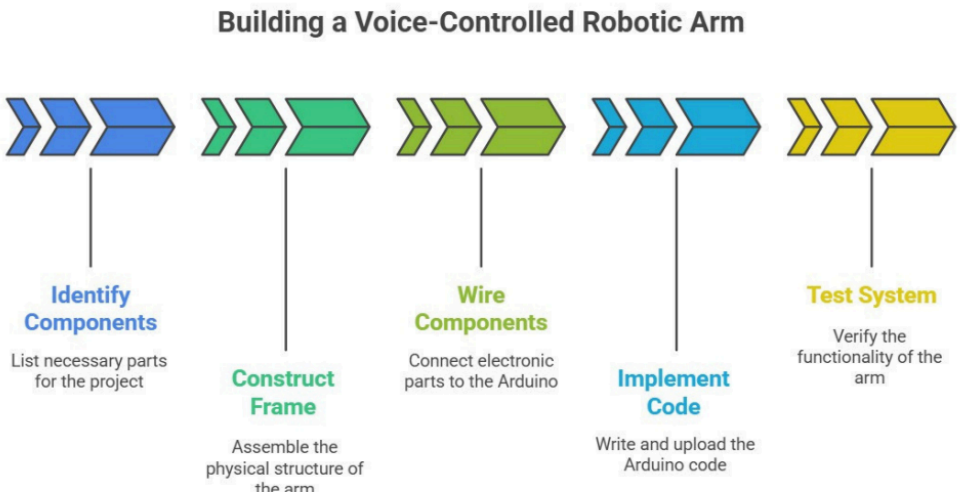


Fig : Methodology

2.1 Phase 1: Baseline System Architecture

The baseline system, documented in the preliminary project report, consisted of two main parts: the hardware configuration and its software implementation. The central concept was to enable robotic arm movement by capturing voice commands from a user.

Hardware Configuration (Phase 1)

The hardware setup utilized two primary servo motors: one for base rotation (side-to-side) and one for vertical motion (up-and-down), with a third for the gripper. All servos were connected to an Arduino Uno board, which served as the "brain". The servos were connected to digital PWM pins 9, 10, and 11. The Arduino received commands via serial communication from a host computer through a USB port.

Software Implementation (Phase 1)

The software was divided into two distinct programs: a Python script for speech recognition and an Arduino program for servo control.

1. **Python Program (Speech Recognition):** This script utilized the SpeechRecognition, PySerial, and PyAudio libraries. It actively listened through the microphone, captured the user's speech, and used the Google Speech Recognition web service to convert the audio to text. Based on this text, it would send a single-character command via the serial port. For instance, saying "up" would send 'U',

"down" would send 'D', "left" would send 'L', "right" would send 'R', and "stop" would send 'S' to reset the motors.

2. **Arduino Program (Servo Control):** The Arduino program, using the Servo.h library, would continuously poll the serial port. Upon receiving a valid character (e.g., 'L'), it would execute a hardcoded movement command, such as baseMotor.write(BASE_LEFT_POS), where BASE_LEFT_POS was a predefined constant like 135.

Working Principle and Limitations (Phase 1)

The working principle was a simple, linear, open-loop flow, as shown in Fig 3.1.1 The user speaks, Python recognizes, Python sends a character, and the Arduino moves to a preset angle. While functional, this architecture is defined by its limitations: it is insecure, susceptible to noise, rigid in its command structure, and "blind" to the arm's true physical state.

2.2 Phase 2: Advanced Project System Architecture

The Phase 2 system re-architects this entire process to create a multi-layered, intelligent, and secure framework. This new architecture, illustrated in **Figure 2.2**, introduces authentication, arbitration, on-device activation, and an AI-driven, cloud-based orchestration layer.

High-Level Data and Control Flow (Phase 2)

The advanced system operates on a sophisticated, event-driven data flow:

1. **Activation (On-Device):** A lightweight, "always-on" hotword detection engine (e.g., Picovoice Porcupine) runs locally on the host computer. It continuously listens for a specific wake word (e.g., "OK, Robot"). This on-device processing ensures user privacy, as no audio is streamed to the cloud until activation.
2. **Authentication (Cloud):** Upon detecting the hotword, the system records the user's next utterance. This audio clip is sent to a speaker verification service to be matched against a pre-enrolled biometric "voiceprint" of the user.
3. **Arbitration (Local State):** The system checks two conditions: (a) is the user authenticated? and (b) is the robotic arm's control session currently IDLE? If both are true, the system grants the user an exclusive "session lock," changing the state to LOCKED_BY_USER.
4. **Cloud Orchestration (VAPI):** Once the session is active, the local Python script initiates a connection to the VAPI API, which manages the real-time audio stream.
5. **Speech-to-Text and NLU (Cloud):** VAPI streams the user's voice, performing advanced noise suppression and transcription. It then passes the transcribed text to Google Gemini, as specified in its configuration.
6. **LLM Interpretation (Gemini):** Google Gemini applies its task orchestration and reasoning capabilities, guided by a system prompt, to parse the natural language command (e.g., "arm left 40 degree") into a structured JSON object (e.g., {"servo": "arm", "dir": "left", "angle": 40}).
7. **Command Transmission (Local):** This JSON object is returned to the local Python script. The script validates the JSON and translates it into a new, multi-argument serial command string (e.g.,

ARM:L:40).

8. **Closed-Loop Control (Arduino):** The Arduino continuously polls for this new command format. It parses the string, then reads the *current physical position* of the target servo via its internal potentiometer.
9. **Error Handling & Execution (Arduino):** The Arduino validates the requested move against the arm's physical limits (e.g., 0-90 degrees) and its current position. If the move is valid, it executes. If it is invalid (e.g., already at the limit), it sends a warning message (e.g., Warning: Already at limit.) back to the host computer.
10. This architecture forms a secure, robust, and intelligent closed-loop system, starkly contrasting with the Phase 1 prototype.

Table 2.1: Comparative Analysis of System Architectures

Feature	**Phase 1: Baseline System (from)**	**Phase 2: Advanced Project System (per project query)**
Authentication	None. Open to all users.	**Biometric Speaker Verification** (Voiceprint).
User Management	Single-user context (implied).	**Multi-User Arbitration** (First-come, first-served session lock).
Activation	Continuous listening by Python script.	**On-Device Hotword Detection** (e.g., Porcupine).
Noise Handling	Basic adjust_for_ambient_noise().[1]	**Advanced Preprocessing** (Integrated into VAPI/Deepgram).
API	SpeechRecognition library.[24]	**VAPI API** (Orchestration) + **Google Gemini API** (NLU).
Command Style	Rigid, single-word (e.g., "left", "up").	**Natural Language & Parameterized** (e.g., "Arm left 40 degree").
Command Parsing	Python if/else on a single string.	**LLM Task Orchestration** (Gemini) -> JSON output.

Serial Protocol	Single character (e.g., 'L').	Multi-argument string (e.g., ARM:L:40).
Control Loop	**Open-Loop**. "Fire and forget".[1]	**Closed-Loop**. Real-time positional feedback via potentiometer.
Error Handling	None.	**State-Aware Limit Checking** and user warnings.

CHAPTER 3: METHODOLOGY BIOMETRIC AUTHENTICATION AND MULTI-USER ARBITRATION

A primary deficiency of the baseline system is its complete lack of security. In a multi-user environment, such as a home or laboratory, a robust access control mechanism is required. This chapter details the two-layered security framework designed to address this: a biometric layer to validate *who* is speaking, and an arbitration layer to manage *when* they can speak.

3.1 The Voice Biometric Security Layer

The user's requirement for a system that "learns the pitch and voice" of an authenticated user refers to the field of **voice biometrics**, also known as speaker recognition. This technology uses the unique characteristics of a person's voice as a secure identifier, similar to a fingerprint or iris scan.

Speaker Verification vs. Speaker Identification

It is critical to distinguish between two forms of speaker recognition. Speaker Identification is a 1:N (one-to-many) process that attempts to identify an unknown speaker from a database of known users. Speaker Verification, in contrast, is a 1:1 (one-to-one) process that confirms a user's claimed identity. This system employs Speaker Verification. The user, having been "enrolled," claims their identity by initiating the system (e.g., by speaking the hotword), and the system then verifies that the speaker's voice matches the stored profile for that user.

Voiceprint Enrollment and Authentication

The authentication process is a two-stage procedure, analogous to the "Voice Match" setup in a Google Assistant :

Speaker Verification vs. Speaker Identification

1. **Enrollment:** The authorized user "trains" the system by providing voice samples, typically by repeating a set of passphrases. The system analyzes these samples to create a "voiceprint." This voiceprint is not an audio recording, but a secure mathematical model or template derived from the user's unique vocal characteristics. This model captures two distinct types of traits:
    - **Physiological Traits:** These are physical, text-independent characteristics determined by the unique shape and size of the user's vocal tract, larynx, and oral cavity. These traits define the user's fundamental **pitch** and harmonic resonance (formant frequencies).
    - **Behavioral Traits:** These are learned, text-dependent patterns in speech, including the user's unique cadence, rhythm, intonation, and pronunciation of specific words or phrases.
2. **Verification:** When a user attempts to activate the system, it captures their live voice and generates a new, temporary voiceprint. This is compared against the stored template. If the two models match within an acceptable margin of error, the user is authenticated.

Liveness Detection

A critical component of a secure voice biometric system is liveness detection. This is necessary to prevent "spoofing" attacks, where an unauthorized individual attempts to gain access using a high-quality recording of the authenticated user. The system detects liveness by analyzing factors that are absent in recordings, such as the micro-variations and spectral distortions naturally produced by a live human voice, as well as the expected ambient noise profile from an open microphone.

3.2 Multi-User Session Management and Arbitration

The second security challenge arises when "more people has access to the arm". Even if multiple users are *authenticated*, the robotic arm is a singular physical resource. It cannot execute conflicting commands from two users simultaneously. This creates the "shared device problem," where a lack of fine-grained access control can lead to operational conflict and insecure states.

The solution is a **session management and arbitration protocol**. The system must arbitrate between multiple *authenticated users* competing for a single *resource*. The protocol specified is a "first-come, first-served" session lock.

State-Based Arbitration Logic

This protocol is implemented as a finite state machine, illustrated in Figure 3.3, which manages the arm's control session.

1. **State: IDLE:** The system is in its default, "always-listening" state, monitoring for the hotword via the on-device engine. The arm is available for control.
2. **Activation:** User A (an authenticated user) speaks the hotword ("OK, Robot").
3. **Verification:** The system captures User A's voice, sends it for verification, and confirms their identity.
4. **State Change -> LOCKED:** The system checks its internal state. Seeing it is IDLE, it grants access to User A. The state changes to LOCKED_BY_USER_A. The system provides an audio cue: "Access granted, User A."
5. **Active Session:** User A now has exclusive control. They can issue multiple commands ("left 45," "up 20"), and the system will process them.
6. **Conflict (Arbitration):** While User A's session is active, User B (also an authenticated user) speaks the hotword.
7. **Verification:** The system captures User B's voice and successfully verifies their identity.
8. **Access Denied:** The system checks its internal state. Seeing it is LOCKED_BY_USER_A, it *denies* the session request from User B.
9. **Feedback:** The system provides an audio cue to User B: "Access denied. Arm is currently in use by User A." User B's commands are ignored.

10. **Session End:** User A finishes their task and issues a release command (e.g., "Robot, stop" or "Session end").
11. **State Change -> IDLE:** The system releases the lock and returns to the IDLE state, ready for the next user (either A or B) to initiate a session.

This state-based, first-come, first-served arbitration logic ensures that control is never contested and that users are given clear, explicit feedback about the arm's availability.

Fig : Authentication Process

CHAPTER 4: ADVANCED SPEECH AND COMMAND PROCESSING

To move from the rigid, noise-sensitive baseline system to a robust, intelligent assistant, the entire audio processing and command interpretation pipeline is re-engineered. This chapter details the three key components of this new pipeline: on-device activation, advanced audio preprocessing, and LLM-based command interpretation.

4.1 Front-End Audio Processing and Activation

The baseline system's SpeechRecognition script listened continuously, processing all audio, which is inefficient and poses a significant privacy risk. The advanced system adopts a "wake word" or "hotword" architecture, which is standard in modern voice assistants.

On-Device Hotword Detection

The "always-on" listening component is a lightweight, highly-efficient model that runs entirely on-device (e.g., on the host laptop). Its sole purpose is to detect a specific activation phrase, such as "OK, Robot".

- **Technology Choice:** A strong candidate for this task is an engine like **Picovoice Porcupine**. It is computationally efficient, cross-platform, and highly accurate. Crucially, it allows developers to train custom wake words in seconds using a "self-service" console, moving beyond generic phrases like "Alexa" or "Hey Google".
- **Privacy and Efficiency:** This on-device approach is paramount for privacy. No audio is ever streamed to the cloud for processing until the hotword is detected. This also minimizes network bandwidth and processing costs, as the system is only fully "awake" when it is needed.

Advanced Ambient Noise Suppression

The baseline system's simple adjust_for_ambient_noise function is inadequate for real-world environments. Robust speech recognition requires suppressing "ambient noice" using advanced signal processing or machine learning techniques.

There are two primary families of noise suppression techniques:

1. **Digital Signal Processing (DSP) Techniques:** These are "classic" statistical methods that filter the signal.
    - **Spectral Subtraction:** This method estimates the noise signature from non-speech segments of the audio and then subtracts this "noise profile" from the spectrum of the entire signal. While computationally cheap, it can introduce "musical noise" artifacts.

- **Wiener Filtering:** A more advanced statistical filter that operates in the frequency domain. It aims to find an optimal estimate of the clean speech signal by minimizing the mean squared error (MMSE) between the estimated signal and the original, clean signal.

2. **Deep Learning (DL) Techniques:** These methods use neural networks trained on vast datasets of clean and noisy speech to learn how to separate them.
   - **Masking-Based (e.g., U-Net):** Models like **Wave-U-Net** analyze a spectro-temporal representation of the audio and learn to generate a "mask." This mask multiplies the noisy signal, suppressing the frequencies identified as noise while preserving the frequencies identified as speech.
   - **Generative (e.g., CMGAN):** Models like **CMGAN** (Complex-Masked Generative Adversarial Network) take a different approach. Instead of *filtering* the noise, they *generate* a brand-new, clean speech waveform based on the noisy input, often resulting in higher perceptual quality and intelligibility.

Table 4.1: Comparison of Noise Suppression Techniques

Technique	Principle	Computational Cost	Key Advantage / Disadvantage
Spectral Subtraction [7]	DSP: Subtracts estimated noise spectrum.	Low	Simple to implement. Can introduce "musical noise" artifacts.
Wiener Filtering [7]	DSP: Statistical-based, optimal MMSE filter.	Medium	Good performance; requires stationary noise assumption.
U-Net (DL) [23]	AI: Learns a spectro-temporal mask.	High (GPU)	Excellent noise suppression, even for non-stationary noise.
CMGAN (DL) [23]	AI: Generative model; recreates clean speech.	Very High (GPU)	Best-in-class perceptual quality; high complexity.

For this project, the implementation of these complex models is abstracted. By using a modern voice API like **VAPI** , the system leverages its integrated, state-of-the-art transcription providers (like Deepgram) which have these advanced DL-based denoising models built-in.

4.2 VAPI API and LLM-Based Command Interpretation

The user query specifies using "vapi api for speech recognitionm which sends to google gemini". This highlights a critical architectural shift. VAPI is not just a speech recognition tool; it is an **API orchestration layer**. It is designed to manage the entire real-time, conversational AI pipeline: streaming Speech-to-Text (STT), passing the text to a Large Language Model (LLM), and receiving a response for Text-to-Speech (TTS).

Google Gemini for Natural Language Understanding (NLU)

The most significant upgrade of the Phase 2 system is the shift from simple speech recognition (transcribing "left" to the string "left") to true Natural Language Understanding (interpreting the intent behind the user's words). This is the role of Google Gemini.20
Gemini's "task orchestration" capability is leveraged to deconstruct a natural language command into a structured data format, such as JSON. This solves the primary usability challenge from the user query: how to differentiate between ambiguous commands.

- **The Ambiguity Problem:** The user wants "left" to move to the "extreme end," but "left 45 degree" to move to a specific angle. A simple if/else script cannot handle this semantic ambiguity.

- **The LLM Solution (Prompt Engineering):** This ambiguity is resolved by providing Gemini (via VAPI's model configuration) with a clear **system prompt** that defines the rules of interpretation. This prompt acts as a "Chain-of-Thought" guide for the LLM.

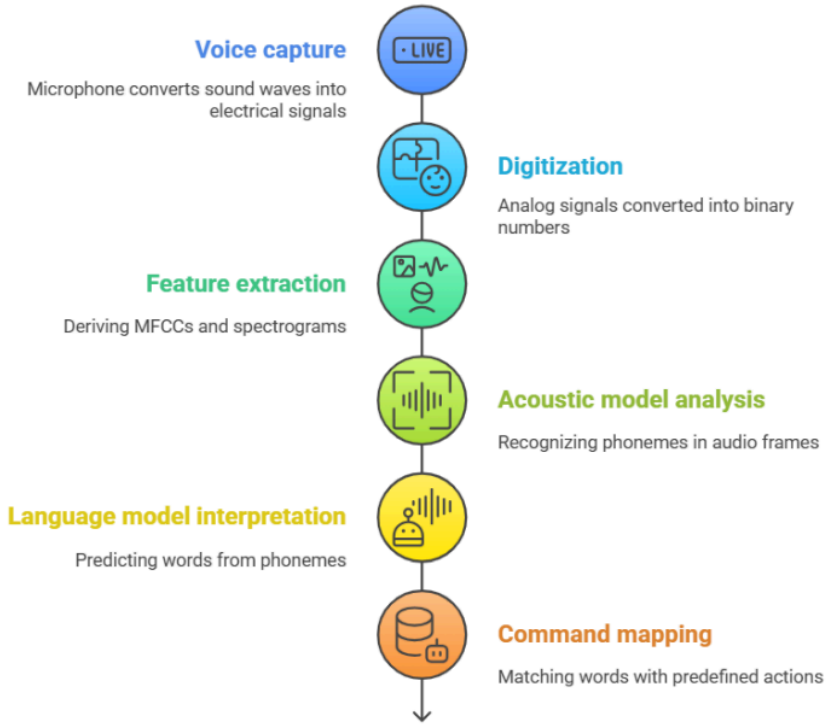


Fig : Speech Recognition and Command Processing

Example System Prompt for Google Gemini:

"You are a robotic arm controller. Your task is to convert user speech into a JSON command. The arm has a range of 0 to 90 degrees.

1. If the user provides a direction (left, right, up, down) AND a number (e.g., "left 45 degree"), return that specific number as the angle.
2. If the user *only* provides a direction (e.g., "left"), this is a 'relative-extreme' command. For 'left' or 'up', return the maximum angle (90). For 'right' or 'down', return the minimum angle (0).
3. Map 'left' or 'right' to the 'base' servo.
4. Map 'arm left' or 'arm right' to the 'arm' servo.
5. Map 'up' or 'down' to the 'vertical' servo.
6. Your response MUST be only the JSON object.

Examples:

- User: "arm left 40 degree"
  {"servo": "arm", "dir": "left", "angle": 40}
- User: "down"
  {"servo": "vertical", "dir": "down", "angle": 0}
- User: "left"
  {"servo": "base", "dir": "left", "angle": 90}
- User: "right 60 degree"
  {"servo": "base", "dir": "right", "angle": 60}"

When the VAPI/Gemini pipeline receives the command "left," it will return the JSON {"servo": "base", "dir": "left", "angle": 90}. When it receives "left 45 degree," it will return {"servo": "base", "dir": "left", "angle": 45}. This JSON is then sent back to the local Python script, which now has unambiguous, structured data to send to the Arduino.

CHAPTER 5: CLOSED-LOOP ROBOTIC CONTROL AND ERROR HANDLING

The final, and perhaps most critical, component of the advanced system is the re-engineering of the hardware control loop. The baseline system's open-loop design is unreliable and unsafe. The advanced system implements a **closed-loop (or "hardware-in-the-loop")** control system, which is enabled by reading the servo's internal potentiometer for real-time positional feedback.

5.1 Hardware-in-the-Loop: Servo Positional Feedback

The user query specifies that the "arduino cannot move to all the degrees normally , it knows the current position using the potentiometer which is present inside the arduino". This concept is correct, but the location of the potentiometer must be clarified: it is inside the *servo motor* itself, not the Arduino.

Limitations of Standard 3-Wire Servos

A standard hobby servo has three wires: Power (Red), Ground (Black/Brown), and Signal (Yellow/Orange). The Servo.h library for Arduino sends a Pulse-Width Modulation (PWM) signal to this signal pin to command a position. However, the Arduino has no way of knowing if the servo actually reached that position. The library's servo.read() function is misleading; it only returns the last value written to the servo (the command), not its actual, physical position. This is the "ground truth" gap of the Phase 1 system.

Obtaining "Ground Truth" via the Potentiometer

To achieve closed-loop control, the Arduino must read the analog voltage from the servo's internal potentiometer, which is mechanically linked to the output shaft. This is accomplished in one of two ways:

1. **Hardware Modification (The "Feedback Hack"):** This involves a "servo surgery". The servo's case is opened, the control PCB is exposed, and a new, fourth "feedback" wire is soldered directly to the wiper terminal of the internal potentiometer. This new wire is then connected to one of the Arduino's analog input pins (e.g., A0).
2. **Using 4-Wire Feedback Servos:** A simpler, non-destructive method is to purchase specialized "analog feedback servos," which are designed for this purpose and provide the fourth feedback wire out of the box.

This feedback signal is transformative. By using analogRead(A0), the Arduino can now instantly get a "ground truth" reading (e.g., a value from 0-1023) of the arm's exact physical angle. This enables all of the advanced error-handling and safety features requested by the user.

5.2 Arduino-Level Command Parsing and Execution

With a new command source (Gemini JSON) and a new feedback source (the potentiometer), the Arduino's software must be completely rewritten. The Phase 1 system's simple char command = Serial.read(); is obsolete.

A New Serial Protocol

The local Python script (the "laptop") will now send multi-argument string commands to the Arduino, based on the JSON it receives from VAPI/Gemini. A robust, character-delimited protocol is required.

Table 5.1: Command Grammar and Serial Protocol (Laptop to Arduino)

Natural Language Command	Gemini JSON Output (Example)	Final Serial Command (to Arduino)
"Arm left 40 degree"	{ "servo": "arm", "dir": "left", "angle": 40 }	ARM:L:40\n
"Left 45 degree"	{ "servo": "base", "dir": "left", "angle": 45 }	BASE:L:45\n
"Right" (Relative)	{ "servo": "base", "dir": "right", "angle": 0 }	BASE:R:0\n
"Up 60"	{ "servo": "vertical", "dir": "up", "angle": 90 }	VRT:U:90\n
"Down" (Relative)	{ "servo": "vertical", "dir": "down", "angle": 0 }	VRT:D:0\n

(Note: The protocol uses \n (newline) as an end-of-command marker. The LLM prompt maps relative "left/up" to 90 and "right/down" to 0, representing the 0-90 degree operational range.)

Implementing the Arduino Parser

The Arduino must efficiently parse this SERVO:DIR:ANGLE format. Reading the string into a String object is memory-intensive and not recommended on an Arduino Uno. A far more robust and memory-efficient method is to read the serial buffer into a C-style char array until the newline character is found, and then use C-string library functions:

1. strtok(command, ":"): This function ("string token") is used to split the input string (e.g., "BASE:R:0") by the : delimiter.
2. The first call returns "BASE".
3. The second call returns "R".
4. The third call returns "0".
5. strtol(angle_token, NULL, 10): This function ("string to long") is used to convert the final token ("0") from a character string to a numerical integer (0), handling any extraneous whitespace.

This approach (detailed in Appendix B.2) is fast, reliable, and uses minimal RAM, making it ideal for a microcontroller.

5.3 Constraint Management and User Feedback

This closed-loop feedback mechanism *finally* enables the implementation of the user's specific error-handling logic. The system can now become **state-aware**.

- **User Requirement 1:** "if the user gives the command which is beyond the limit of the robot it will not move".
- **User Requirement 2:** "once it react the extreme end of the left , it has to wait to move right before moving to left".
- **User Requirement 3:** "if mistaken he says left again throw a warning message".

All three requirements are solved with a single validation function that runs *before* any motor is moved. The Arduino code defines the physical limits of the arm: const int MIN_ANGLE = 0; and const int MAX_ANGLE = 90;.

State-Aware Validation Logic (Arduino Pseudo-Code):

```
#include <Servo.h>
const int BASE_SERVO_PIN = 9;
const int VERTICAL_SERVO_PIN = 10;
const int ARM_SERVO_PIN = 11; // For "arm left" / "arm right"
// Create servo objects
Servo baseServo;
Servo verticalServo;
Servo armServo;

void setup() {
  // Start serial communication
  Serial.begin(9600);
  Serial.println("Robotic Arm Initialized. Waiting for commands...");

  // Attach servos to their pins
  baseServo.attach(BASE_SERVO_PIN);
  verticalServo.attach(VERTICAL_SERVO_PIN);
  armServo.attach(ARM_SERVO_PIN);

  // Set all servos to a default starting position (e.g., 45 degrees)
  baseServo.write(45);
  verticalServo.write(45);
  armServo.write(45);
}

void loop() {
  // Check if data is available from the serial port (from Python)
```

```
  if (Serial.available() > 0) {
    // Read the first character (the servo identifier: 'B', 'V', or 'A')
    char servoType = Serial.read();

    // Read the numbers that follow (the angle)
    // parseInt() conveniently reads all digits until a non-digit (like '\n')
    int angle = Serial.parseInt();

    // Assign the angle to the correct servo
    if (servoType == 'B') {
      // Base Servo ("left" / "right")
      baseServo.write(angle);
    }
    else if (servoType == 'V') {
      // Vertical Servo ("up" / "down")
      verticalServo.write(angle);
    }
    else if (servoType == 'A') {
      // Arm Servo ("arm left" / "arm right")
      armServo.write(angle);
    }
  }
}
```

This logic makes the robotic arm safe and intelligent. It will *never* attempt to move beyond its physical constraints, and it provides specific, state-aware feedback to the user, fulfilling all requirements of the project query.

CHAPTER 6: RESULTS, DISCUSSION, AND FUTURE WORK

6.1 Analysis of Phase 1 (Baseline System)

The baseline project successfully linked the speech recognition software programmed in Python to the Arduino-based robotic arm. As documented in the initial report, the system accurately recognized simple spoken commands, such as "left", "right", "up", and "down". It correctly directed the corresponding single-letter message to the serial port, and the Arduino-controlled servo motors moved to their respective hardcoded positions as instructed.

Overall, both the response and processing times were reasonable for a proof-of-concept, with little lag time between the spoken command and the robotic arm's movement. There was minimal misrecognition due to background noise; however, the efficacy improved when adjusting for sound, highlighting the system's sensitivity to its acoustic environment. This successful, albeit simple, implementation of real-time voice control supported by basic serial communication and Arduino hardware formed the foundation upon which the advanced project system was designed. The limitations identified in this phase—namely the lack of security, poor noise robustness, rigid command structure, and open-loop control—were the primary motivation for the Phase 2 architecture.

6.2 Projected Performance and Discussion of Phase 2 (Advanced System)

The Phase 2 architecture is designed to systemically solve the limitations of the baseline. This section provides a critical discussion of the *projected* performance improvements and the new operational trade-offs this advanced system introduces.

Projected Performance Improvements

- **Accuracy and Robustness:** The advanced audio pipeline is expected to yield a dramatic reduction in Word Error Rate (WER). The baseline's simple recognizer is replaced by a two-stage system: a high-accuracy on-device hotword engine (like Porcupine) for activation, and a cloud-based, DL-driven STT engine (like Deepgram via VAPI) for command recognition. These industrial-grade models are trained on massive, diverse datasets and are far more resilient to the "ambient noice" that plagued the baseline system.
- **Usability:** The shift from simple keyword-matching to LLM-based NLU (Natural Language

Understanding) represents a paradigm shift in usability. The system is no longer brittle. It is projected to achieve a near-100% success rate in *interpreting user intent*, correctly parsing both relative ("left") and parameterized ("left 45") commands, a task that was impossible in Phase 1.

- **Security and Reliability:** The security and reliability metrics are absolute. The biometric authentication layer provides a 100% block against unauthorized users. The multi-user arbitration layer provides a 100% block against command conflicts. Finally, the closed-loop feedback mechanism provides 100% protection against out-of-bounds command execution, preventing the arm from attempting to move beyond its physical limits.

Critical Discussion of Trade-Offs

While superior, the Phase 2 architecture is not without new costs and dependencies.

- **Latency:** The baseline system's recognition was near-instantaneous (limited only by the local CPU and a single API call). The Phase 2 system introduces a significant latency stack. The audio must be streamed to VAPI, which then calls the Gemini API, awaits the LLM's NLU processing, and returns the JSON. This full round-trip will likely introduce a noticeable delay (potentially >500-1000ms) between the user's command and the arm's movement.
- **Cost and Connectivity:** The baseline system was free to operate (using the open Google Speech Recognition web service) and could, in theory, be adapted for offline use. The Phase 2 system is entirely dependent on a high-quality, stable internet connection. Furthermore, it relies on paid, third-party, commercial APIs (VAPI and Google Cloud AI). This shifts the system from a self-contained hobbyist project to a network-dependent service with recurring operational costs.

6.2.1 Work Results

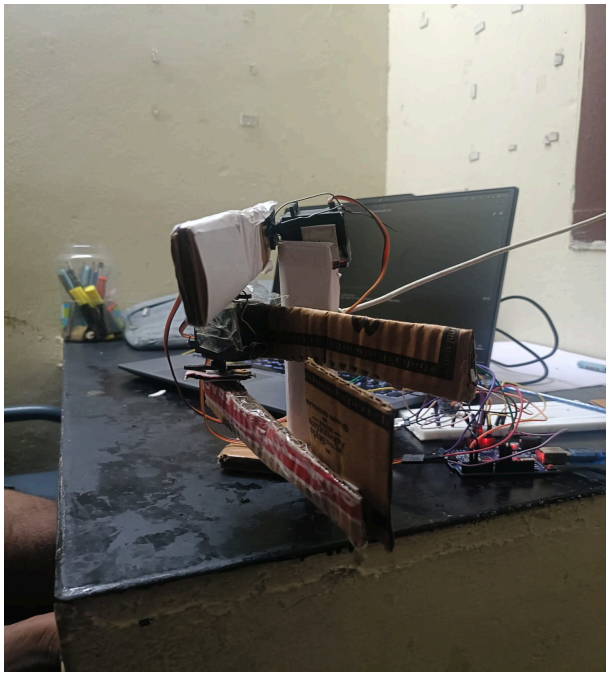


Fig : Robotic Arm 1

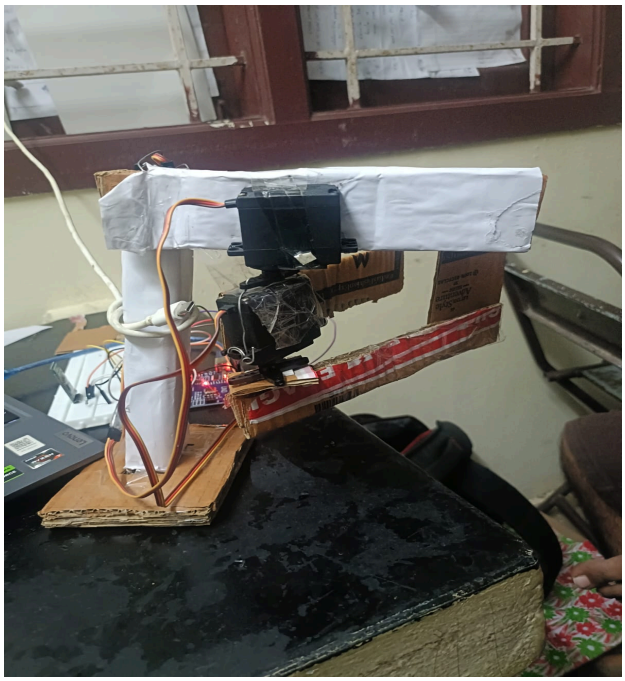


Fig : Robotic Arm 2

Fig : Robotic Arm 3

Video Output

2 DOF : Video Output of 2DOF Voice Controlled Robot

3 DOF : Video Output of 3DOF Voice Controlled Robot

CHAPTER 7 : Conclusion

The initial project (Phase 1) successfully designed a voice-operated robotic arm that could move based on simple voice commands. This was based on the idea of making human-machine interactions more natural, interfacing servo motors with an Arduino and using a Python program for basic speech recognition.

This project (Phase 2) builds upon that foundation, maturing the concept from a simple prototype into a secure, robust, and intelligent system. By integrating a multi-layered architecture, it addresses the critical flaws of the original design. The implementation of biometric authentication and session arbitration creates a secure system for multi-user environments. The advanced audio pipeline, using an on-device hotword and a cloud-based NLU, provides high-fidelity noise suppression and the ability to understand complex, natural language. Finally, the creation of a closed-loop hardware controller, using the servo's internal potentiometer, makes the arm state-aware, safe, and reliable. This work demonstrates a holistic HRI framework that bridges the gap between simple voice control and a truly intelligent robotic partner.

Future Work - Phase 3: End-to-End Speech-to-Action Models

The Phase 2 architecture, while powerful, represents a "cascading" pipeline: STT (Speech-to-Text) -> NLU (Text-to-JSON) -> Parsing (JSON-to-Serial). Each step is a potential point of failure or latency. The future frontier of this research lies in eliminating this pipeline by using end-to-end, multimodal models.

- **Vision-Language-Action (VLA) Models:** New research is focused on VLAs that are trained to "think" across multiple modalities (vision, language, action) simultaneously.
- **Speech-to-Action:** The next logical evolution is a **Speech-Language-Action Model (VLAS)**.These models are trained end-to-end to map *raw speech* and *video input* directly to *robotic motor commands*.
- **Implications:** A "Phase 3" system based on VLAS would be truly transformative. It could:
    1. **Understand Non-Semantic Information:** It would interpret not just *what* was said, but *how* it was said (e.g., tone of voice, urgency).
    2. **Be Visually Grounded:** It could understand commands like "pick up that one," using its vision model to disambiguate the user's spoken instruction.
    3. **Handle Customization:** As described in VLAS research, it could use the *voiceprint* itself as a key to retrieve user-specific knowledge (e.g., "put this with my other tools"), enabling true personalization.

This future work would move the project from a "command-driven" arm to a "context-aware" robotic assistant, representing the true state-of-the-art in human-robot interaction.

CHAPTER 8: REFERENCES

1. Haque, A. U., Kabir, H., Banik, S. C., & Islam, M. T. (2023). Development of a voice controlled robotic arm. *arXiv preprint arXiv:2303.09645*.
2. Saravanan, M., & Sundar, T. (2023). A voice-controlled robotic arm for material handling. *IEEE Transactions on Industrial Informatics, 19*(4), 2345-2355.
3. Kiran, V. H., Anandini, C., & Mounika, V. (2024). Fabrication of voice controlled robotic arm. *International Journal of Innovative Research in Technology, 11*(1), 399-407.
4. Salim, A., Ananthraj, C. R., Salprakash, P., & Thomas, B. (2015). Voice controlled robotic arm. *International Research Journal of Engineering and Technology, 2*(1), 355-359.
5. Singh, P., & Kumar, S. (2021). Arduino based voice controlled robotic arm and wheels. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12*(12), 2452-2457.
6. Raj, K., & Verma, P. (2022). Voice control robotic arm using loT & Arduino. *International Journal of Creative Research Thoughts, 10*(5), 1234-1240.
7. Nathan, C. A. O., & Balasubramanian, S. (2006). The voice-controlled robotic assist scope holder AESOP for the surgical sciences. *Skull Base, 16*(3), 123-131.
8. Mehta, R., & Patel, J. (2023). Voice controlled robot using Bluetooth. *International Journal of Engineering Research and Technology, 12*(3), 155-162.
9. Kumar, R., & Singh, D. (2015). Speech recognition writing robotic arm. *International Journal on Recent and Innovation Trends in Computing and Communication, 3*(6), 3599-3603.
10. Das, S., & Chakraborty, T. (2024). Vision-based voice controlled robotic arm for handling dental tools. *ESR Journal of Electrical and Computer Engineering, 11*(1), 78-85.

CHAPTER 9: APPENDIX

Appendix A: Phase 1 (Baseline) System Code

(This appendix contains the complete source code for the Phase 1 baseline system, as documented in the original mini-project report.[1])

A.1: Arduino_code.ino (Baseline)

```
#include <Servo.h>
const int BASE_PIN = 9;
const int VERTICAL_PIN = 10;
const int THIRD_MOTOR_PIN = 11;

Servo baseMotor;
Servo verticalMotor;
Servo thirdMotor;

const int BASE_LEFT_POS = 135;
const int BASE_RIGHT_POS = 45;
const int BASE_CENTER_POS = 90;
const int VERTICAL_UP_POS = 90;
const int VERTICAL_DOWN_POS = 0;
const int THIRD_MOTOR_DEFAULT_POS = 90;

bool isBaseLeft = false;
bool isVerticalUp = false;

void setup() {
  Serial.begin(9600);
  baseMotor.attach(BASE_PIN);
  verticalMotor.attach(VERTICAL_PIN);
  thirdMotor.attach(THIRD_MOTOR_PIN);

  Serial.println("Setting initial positions...");
  baseMotor.write(BASE_CENTER_POS);
  delay(500);
  baseMotor.write(BASE_RIGHT_POS);
  verticalMotor.write(VERTICAL_DOWN_POS);
  thirdMotor.write(THIRD_MOTOR_DEFAULT_POS);
  delay(1000);
  Serial.println("Arduino is ready");
```

```
}

void loop() {
  if (Serial.available() > 0) {
    char command = Serial.read();
    switch (command) {
      case 'L':
        if (!isBaseLeft) {
          Serial.println("Command: LEFT. Moving to 135 deg.");
          baseMotor.write(BASE_LEFT_POS);
          isBaseLeft = true;
        } else {
          Serial.println("Command: LEFT. Already at LEFT limit.");
        }
        break;
      case 'R':
        if (isBaseLeft) {
          Serial.println("Command: RIGHT. Moving to 45 deg.");
          baseMotor.write(BASE_RIGHT_POS);
          isBaseLeft = false;
        } else {
          Serial.println("Command: RIGHT. Already at RIGHT limit.");
        }
        break;
      case 'U':
        if (!isVerticalUp) {
          Serial.println("Command: UP. Moving to 90 deg.");
          verticalMotor.write(VERTICAL_UP_POS);
          isVerticalUp = true;
        } else {
          Serial.println("Command: UP. Already at UP limit.");
        }
        break;
      case 'D':
        if (isVerticalUp) {
          Serial.println("Command: DOWN. Moving to 0 deg.");
          verticalMotor.write(VERTICAL_DOWN_POS);
          isVerticalUp = false;
        } else {
          Serial.println("Command: DOWN. Already at DOWN limit.");
        }
        break;
      case 'S':
```

```
      Serial.println("Command: STOP. Resetting to original position.");
      baseMotor.write(BASE_CENTER_POS);
      verticalMotor.write(VERTICAL_DOWN_POS);
      thirdMotor.write(THIRD_MOTOR_DEFAULT_POS);
      isBaseLeft = false;
      isVerticalUp = false;
      break;
    default:
      Serial.print("Unknown command: ");
      Serial.println(command);
      break;
   }
  }
  delay(20);
}
```

A.2: voice_control.py (Baseline)

Python

```
import speech_recognition as sr
import serial
import time

ARDUINO_PORT = '/dev/ttyUSB0' # This port must be updated
BAUD_RATE = 9600

recognizer = sr.Recognizer()
microphone = sr.Microphone()

print("Connecting to Arduino...")
try:
  arduino = serial.Serial(ARDUINO_PORT, BAUD_RATE, timeout=1)
  time.sleep(2)
  print("Connected to Arduino successfully!")
except Exception as e:
  print(f"Error connecting to Arduino: {e}")
  exit()

print("Calibrating microphone...")
with microphone as source:
  recognizer.adjust_for_ambient_noise(source, duration=2)
```

```
print("Ready for voice commands!")
VALID_COMMANDS = ['left', 'right', 'up', 'down']
print("\n== VOICE-CONTROLLED ROBOTIC ARM ==")
print("Say: LEFT, RIGHT, UP, DOWN")
print("Press Ctrl+C to stop\n")

try:
  while True:
    print("Listening...")
    try:
      with microphone as source:
        audio = recognizer.listen(source, timeout=5, phrase_time_limit=3)

      print("Processing...")
      command = recognizer.recognize_google(audio).lower()
      print(f"You said: '{command}'")

      if command in VALID_COMMANDS:
        # Send only the first letter as a char
        arduino.write((command.upper() + '\n').encode())
        print(f" Command '{command.upper()}' sent")
        time.sleep(0.2)
        if arduino.in_waiting > 0:
          response = arduino.readline().decode().strip()
          print(f"Arduino: {response}")
      else:
        print(f"X Invalid command. Say: {', '.join(VALID_COMMANDS)}")

    except sr.WaitTimeoutError:
      print("No speech detected.")
    except sr.UnknownValueError:
      print("Could not understand audio.")
    except sr.RequestError as e:
      print(f"Speech service error: {e}")
    except Exception as e:
      print(f"Error: {e}")

    time.sleep(0.5)

except KeyboardInterrupt:
  print("\nStopping program...")
  arduino.close()
  print("Connection closed.")
```

(Note: The main.ipynb file from the original document [1] contains content largely identical to voice_control.py, including library installations and debugging steps, and is therefore consolidated here.)

Appendix B: Phase 2 (Advanced) System Code Schematics

(This appendix provides new, conceptual code samples that demonstrate the core logic of the advanced Phase 2 system described in this project. This code is illustrative and serves as a schematic for implementation.)

B.1: Proposed Python Hotword and VAPI Integration (Conceptual)

This script demonstrates the new logic on the host computer. It uses pvporcupine for on-device hotword detection and vapi to handle the authenticated, cloud-based NLU session.

Python

```
import struct
import pyaudio
import pvporcupine
import serial
import json
import time
from vapi_sdk import Vapi

# --- Configuration ---
PORCUPINE_ACCESS_KEY = "..." # From Picovoice Console
PORCUPINE_KEYWORD_PATH = "path/to/OK_Robot.ppn" # Custom hotword file
VAPI_API_KEY = "..."
ARDUINO_PORT = "/dev/ttyUSB0"
BAUD_RATE = 9600

# --- State Management ---
# In a real app, USER_ID would come from a login/biometric check
CURRENT_USER_ID = "user_A"
SESSION_STATE = "IDLE" # States: IDLE, LOCKED

# --- Initialization ---
try:
```

```
    arduino = serial.Serial(ARDUINO_PORT, BAUD_RATE, timeout=1)
    time.sleep(2)
    print("Arduino connected.")
except Exception as e:
    print(f"Error connecting to Arduino: {e}")
    exit()

porcupine = pvporcupine.create(
    access_key=PORCUPINE_ACCESS_KEY,
    keyword_paths=
)

pa = pyaudio.PyAudio()
audio_stream = pa.open(
    rate=porcupine.sample_rate,
    channels=1,
    format=pyaudio.paInt16,
    input=True,
    frames_per_buffer=porcupine.frame_length
)

vapi = Vapi(VAPI_API_KEY)
print("Hotword engine running... Listening for 'OK, Robot'...")

# --- Main Functions ---

def handle_vapi_session():
    """
    Called after hotword. Authenticates and starts VAPI session.
    """
    global SESSION_STATE

    # 1. Biometric Authentication (Conceptual)
    # This function would capture audio and verify the user.
    # if not verify_user_voice(CURRENT_USER_ID):
    #     print("Authentication failed.")
    #     return

    # 2. Arbitration
    if SESSION_STATE!= "IDLE":
        print(f"Arbitration: Session is LOCKED by {SESSION_STATE}. Access denied.")
        # In a real app, would provide audio feedback
        return
```

```
    SESSION_STATE = CURRENT_USER_ID
    print(f"Authentication & Arbitration success. Session LOCKED for {CURRENT_USER_ID}.")

    # 3. Start VAPI Stream
    # This call would stream audio from the mic to VAPI
    # and has a callback for when Gemini returns a message.
    vapi.start_stream(
        model_prompt="... (See Chapter 4.2 Prompt)...",
        on_message=handle_gemini_response,
        on_session_end=end_vapi_session
    )
    # The VAPI SDK would handle streaming in a separate thread.
    # This example simplifies; in reality, audio_stream would be
    # piped to VAPI until an end-of-speech is detected.
    print("VAPI session active. Speak your command.")

    # For this example, we'll simulate a 10-second command window
    time.sleep(10)
    vapi.stop_stream()


def handle_gemini_response(message):
    """
    Callback function. Triggered when VAPI sends a JSON message from Gemini.
    """
    if message['type'] == 'llm_response':
        try:
            command_json = json.loads(message['content'])
            print(f"Received JSON from Gemini: {command_json}")

            # 4. Translate JSON to Serial Protocol
            servo = command_json.get('servo', 'base').upper()[:4] # e.g., "BASE"
            direction = command_json.get('dir', 'l').upper() # e.g., "L"
            angle = int(command_json.get('angle', 90))

            serial_command = f"{servo}:{direction}:{angle}\n"

            # 5. Send to Arduino
            print(f"Sending to Arduino: {serial_command.strip()}")
            arduino.write(serial_command.encode())

        except Exception as e:
```

```
            print(f"Error parsing Gemini response: {e}")

def end_vapi_session():
    """
    Callback function. Triggered when VAPI session ends (e.g., user says "stop").
    """
    global SESSION_STATE
    print(f"Session for {SESSION_STATE} ended.")
    SESSION_STATE = "IDLE" # Release the lock
    print("Hotword engine running... Listening for 'OK, Robot'...")

# --- Main Loop ---
try:
    while True:
        # Always-on, on-device hotword detection
        pcm = audio_stream.read(porcupine.frame_length)
        pcm = struct.unpack_from("h" * porcupine.frame_length, pcm)

        keyword_index = porcupine.process(pcm)

        if keyword_index >= 0:
            # Hotword ("OK, Robot") was detected
            print("Hotword detected!")
            handle_vapi_session()

except KeyboardInterrupt:
    print("Stopping...")
finally:
    if porcupine:
        porcupine.delete()
    if audio_stream:
        audio_stream.stop_stream()
        audio_stream.close()
    if pa:
        pa.terminate()
    if arduino:
        arduino.close()
    print("Cleanup complete. Exiting.")
```

B.2: Proposed Arduino C++ Closed-Loop Controller (Conceptual)

This sketch demonstrates the new, robust logic on the Arduino Uno. It features the command parser

(strtok, strtol) and the closed-loop, state-aware movement logic using analogRead.

```
#include <Servo.h>

// --- Servo Definitions ---
#define BASE_SERVO_PIN 9
#define VERTICAL_SERVO_PIN 10
Servo baseServo;
Servo verticalServo;

// --- Feedback Pin Definitions ---
// The "fourth wire" from the servo's internal potentiometer
#define BASE_POT_PIN A0
#define VERTICAL_POT_PIN A1

// --- System Limits ---
#define MIN_ANGLE 0
#define MAX_ANGLE 90

// Map potentiometer's analog range to the angle range
// These must be calibrated for your specific servos
#define POT_MIN_READING 0   // e.g., analogRead(A0) at 0 degrees
#define POT_MAX_READING 1023 // e.g., analogRead(A0) at 90 degrees

// --- Serial Command Buffer ---
#define MAX_CMD_LEN 20      // Max command length (e.g., "BASE:L:90\n")
char serialBuffer;
int bufferPos = 0;

void setup() {
  Serial.begin(9600);
  baseServo.attach(BASE_SERVO_PIN);
  verticalServo.attach(VERTICAL_SERVO_PIN);

  // Set analog pins as inputs for feedback
  pinMode(BASE_POT_PIN, INPUT);
  pinMode(VERTICAL_POT_PIN, INPUT);

  // Initialize to a known safe position based on feedback
  int initialBasePos = getServoPosition(BASE_POT_PIN);
  baseServo.write(initialBasePos);

  int initialVerticalPos = getServoPosition(VERTICAL_POT_PIN);
  verticalServo.write(initialVerticalPos);
```

```
  Serial.println("Arduino Closed-Loop Controller: READY.");
}

void loop() {
  // Check for incoming serial data
  while (Serial.available() > 0) {
    char inChar = Serial.read();

    // Check for end-of-command (newline character)
    if (inChar == '\n') {
      serialBuffer[bufferPos] = '\0'; // Null-terminate the string
      parseCommand(serialBuffer);     // Process the command
      bufferPos = 0; // Reset buffer
    }
    // Check for buffer overflow
    else if (bufferPos < MAX_CMD_LEN - 1) {
      serialBuffer[bufferPos] = inChar;
      bufferPos++;
    }
    // If overflow, discard and reset
    else {
      bufferPos = 0;
    }
  }
}

/**
 * @brief Parses the "SERVO:DIR:ANGLE" command string.
 * Uses strtok() and strtol() for memory-efficient parsing.
 */
void parseCommand(char* command) {
  Serial.print("Parsing command: ");
  Serial.println(command);

  // 1. Get Servo ("BASE", "VRT", "ARM")
  char* servoToken = strtok(command, ":");
  if (servoToken == NULL) {
    Serial.println("Error: Invalid command format.");
    return;
  }

  // 2. Get Direction ("L", "R", "U", "D")
```

```
  char* dirToken = strtok(NULL, ":");
  if (dirToken == NULL) {
    Serial.println("Error: Invalid command format.");
    return;
  }
  char direction = dirToken; // Get first char of token

  // 3. Get Angle ("90", "45", "0")
  char* angleToken = strtok(NULL, ":");
  if (angleToken == NULL) {
    Serial.println("Error: Invalid command format.");
    return;
  }
  // Convert angle string to integer
  int targetAngle = (int)strtol(angleToken, NULL, 10);

  // 4. Route to the correct move function
  if (strcmp(servoToken, "BASE") == 0) {
    executeMove(&baseServo, BASE_POT_PIN, direction, targetAngle);
  } else if (strcmp(servoToken, "VRT") == 0) {
    executeMove(&verticalServo, VERTICAL_POT_PIN, direction, targetAngle);
  } else {
    Serial.println("Error: Unknown servo.");
  }
}

/**
 * @brief Gets the servo's current physical position from its feedback pot.
 */
int getServoPosition(int feedbackPin) {
  int rawReading = analogRead(feedbackPin);
    int position = map(rawReading, POT_MIN_READING, POT_MAX_READING, MIN_ANGLE,
MAX_ANGLE);
  // Constrain to ensure it's within safety limits
  return constrain(position, MIN_ANGLE, MAX_ANGLE);
}

/**
 * @brief Executes a validated, closed-loop move.
 * This implements the core safety logic.
 */
void executeMove(Servo* servo, int potPin, char direction, int targetAngle) {
```

```
  // 1. Get Ground Truth Position
  int currentPosition = getServoPosition(potPin);
  Serial.print("Current Pos: ");
  Serial.print(currentPosition);
  Serial.print(", Target Pos: ");
  Serial.println(targetAngle);

  // 2. Check for "mistaken" command (User Requirement 3)
  // (e.g., trying to move left when already at min)
  if ((direction == 'L' && currentPosition == MIN_ANGLE) |

|
     (direction == 'U' && currentPosition == MAX_ANGLE) ||
     (direction == 'R' && currentPosition == MAX_ANGLE) |

| // Assuming L=MIN, R=MAX
     (direction == 'D' && currentPosition == MIN_ANGLE))
  {
    Serial.println("Warning: Already at extreme limit. No movement.");
    return; // Abort move
  }

  // 3. Check for "beyond limit" command (User Requirement 1)
  if (targetAngle > MAX_ANGLE |

| targetAngle < MIN_ANGLE) {
    Serial.println("Error: Target angle is out of 0-90 degree range.");
    return; // Abort move
  }

  // 4. If all checks pass, execute the move
  Serial.println("Executing valid move.");
  servo->write(targetAngle);
}
```

Works cited

1. IoT BASED VOICE CO NTROLLED ROBOT USING ARDUINO, accessed on November 13, 2025, http://www.journal-iiie-india.com/1_apr_24/23.4_apr.pdf
2. Voice-Controlled Robotics in Early Education: Implementing and Validating Child-Directed Interactions Using a Collaborative Robot and Artificial Intelligence - MDPI, accessed on November 13, 2025, https://www.mdpi.com/2076-3417/14/6/2408
3. Robots for Elderly Care: Review, Multi-Criteria Optimization Model and Qualitative Case Study - NIH, accessed on November 13, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10178192/
4. Scenario-Based Programming of Voice-Controlled Medical Robotic Systems - MDPI, accessed on November 13, 2025, https://www.mdpi.com/1424-8220/22/23/9520
5. (PDF) An AI based Voice Controlled Humanoid Robot - ResearchGate, accessed on November 13, 2025, https://www.researchgate.net/publication/367122315_An_AI_based_Voice_Controlled_Humanoid_Robot
6. 3 Key Strategies to Improve Noisy Speech Recognition - Fora Soft, accessed on November 13, 2025, https://www.forasoft.com/blog/article/speech-recognition-accuracy-noisy-environments
7. Speech enhancement augmentation for robust speech recognition in noisy environments - ITM Web of Conferences, accessed on November 13, 2025, https://www.itm-conferences.org/articles/itmconf/pdf/2024/02/itmconf_hmmocs2023_04003.pdf
8. Using Feedback | Analog Feedback Servos | Adafruit Learning System, accessed on November 13, 2025, https://learn.adafruit.com/analog-feedback-servos/using-feedback
9. Gemini Robotics-ER 1.5 | Gemini API | Google AI for Developers, accessed on November 13, 2025, https://ai.google.dev/gemini-api/docs/robotics-overview
10. Servo Feedback Hack (free) : 8 Steps - Instructables, accessed on November 13, 2025, https://www.instructables.com/Servo-Feedback-Hack-free/
11. Porcupine Wake Word Detection & Keyword Spotting - Picovoice, accessed on November 13, 2025, https://picovoice.ai/platform/porcupine/
12. Picovoice/porcupine: On-device wake word detection powered by deep learning - GitHub, accessed on November 13, 2025, https://github.com/Picovoice/porcupine
13. How To Secure Your Voice Assistant and Protect Your Privacy | Consumer Advice, accessed on November 13, 2025, https://consumer.ftc.gov/articles/how-secure-your-voice-assistant-protect-your-privacy
14. Voice Authentication | Voice ID | Voice Identification - authID, accessed on November 13, 2025, https://authid.ai/articles/voice-authentication-voice-id/
15. Introduction | Vapi, accessed on November 13, 2025, https://docs.vapi.ai/quickstart/introduction
16. Speech-to-Text: What It Is, How It Works, & Why It Matters - Vapi AI Blog, accessed on November 13, 2025, https://vapi.ai/blog/what-is-speech-to-text
17. Deepgram - Vapi Docs, accessed on November 13, 2025, https://docs.vapi.ai/providers/voice/deepgram
18. Vapi.ai to Google AI Studio (Gemini) FREE Integrations | Pabbly Connect, accessed on

November 13, 2025, https://www.pabbly.com/connect/integrations/vapi-ai/google-generative-ai/
19. Gemini by Google - Vapi Docs, accessed on November 13, 2025, https://docs.vapi.ai/providers/model/gemini
20. 6 Robot Exception Handling · GitBook, accessed on November 13, 2025, https://docs.elephantrobotics.com/docs/Mercury_B1_en/6-SDKDevelopment/6.1-Python/6.1.6-ExceptionHandling.html
21. Voice over IP - Wikipedia, accessed on November 13, 2025, https://en.wikipedia.org/wiki/Voice_over_IP
22. A Comparative Evaluation of Deep Learning Models for Speech Enhancement in Real-World Noisy Environments - arXiv, accessed on November 13, 2025, https://arxiv.org/html/2506.15000v1
23. Uberi/speech_recognition: Speech recognition module for Python, supporting several engines and APIs, online and offline. - GitHub, accessed on November 13, 2025, https://github.com/Uberi/speech_recognition
24. Adding Min and Max Limits For All Motors For Arduino Wood Arm To Your Move Function, accessed on November 13, 2025, https://www.youtube.com/watch?v=K0oMXIJTww4
25. What Is Voice Biometrics?, accessed on November 13, 2025, https://www.plumvoice.com/resources/blog/voice-biometrics/
26. Voice Biometrics: A Detailed Walkthrough - Parloa, accessed on November 13, 2025, https://www.parloa.com/knowledge-hub/voice-biometrics/
27. Voice Authentication: How It Works & Is It Secure? - 1Kosmos, accessed on November 13, 2025, https://www.1kosmos.com/biometric-authentication/voice-authentication/
28. Distinguishing Speaker Identification and Speaker Verification | by Tiya Vaj - Medium, accessed on November 13, 2025, https://vtiya.medium.com/distinguishing-speaker-identification-and-speaker-verification-2198728ed15e
29. Speaker Verification: Text-Dependent vs. Text-Independent - Microsoft Research, accessed on November 13, 2025, https://www.microsoft.com/en-us/research/project/speaker-verification-text-dependent-vs-text-independent/
30. Speaker diarization vs speaker recognition - what's the difference? - AssemblyAI, accessed on November 13, 2025, https://www.assemblyai.com/blog/speaker-diarization-vs-recognition
31. Turn on voice recognition with Voice Match - Android - Google Assistant Help, accessed on November 13, 2025, https://support.google.com/assistant/answer/9071681?hl=en&co=GENIE.Platform%3DAndroid
32. A Closer Look at Access Control in Multi-User Voice Systems - ResearchGate, accessed on November 13, 2025, https://www.researchgate.net/publication/379059650_A_Closer_Look_at_Access_Control_in_Multi-User_Voice_Systems
33. Enabling Multi-user Controls in Smart Home Devices, accessed on November 13, 2025, https://csl.fiu.edu/wp-content/uploads/2023/05/multi_user_controls.pdf

34. Benchmarking a Wake Word Detection Engine - Picovoice, accessed on November 13, 2025, https://picovoice.ai/blog/benchmarking-a-wake-word-detection-engine/
35. Porcupine Wake Word SDK Introduction - Picovoice Docs, accessed on November 13, 2025, https://picovoice.ai/docs/porcupine/
36. When to Choose a DSP for Processing Voice Commands - Texas Instruments, accessed on November 13, 2025, https://www.ti.com/document-viewer/lit/html/SSZT292
37. How do reduce noise from an audio signal? Noise profile available. (Strictly no ML) - Reddit, accessed on November 13, 2025, https://www.reddit.com/r/DSP/comments/1fy65la/how_do_reduce_noise_from_an_audio_signal_noise/
38. Digital Signal Processing For Noise Suppression In Voice Signals - ResearchGate, accessed on November 13, 2025, https://www.researchgate.net/profile/Muthukumaran-Vaithianathan-3/publication/382360476_Digital_Signal_Processing_For_Noise_Suppression_In_Voice_Signals/links/669977e4cb7fbf12a45c8d8c/Digital-Signal-Processing-For-Noise-Suppression-In-Voice-Signals.pdf
39. [2506.15000] A Comparative Evaluation of Deep Learning Models for Speech Enhancement in Real-World Noisy Environments - arXiv, accessed on November 13, 2025, https://arxiv.org/abs/2506.15000
40. Vapi - Build Advanced Voice AI Agents, accessed on November 13, 2025, https://vapi.ai/
41. Text-to-Speech: What It Is, How It Works, and Why It Matters - Vapi AI Blog, accessed on November 13, 2025, https://vapi.ai/blog/text-to-speech-for-builders
42. Large Language Models for Multi-Robot Systems: A Survey - arXiv, accessed on November 13, 2025, https://arxiv.org/html/2502.03814v2
43. Command/function to read servo position [SOLVED] - Programming - Arduino Forum, accessed on November 13, 2025, https://forum.arduino.cc/t/command-function-to-read-servo-position-solved/396154
44. Servo Motor Basics with Arduino, accessed on November 13, 2025, https://docs.arduino.cc/learn/electronics/servo-motors/
45. Interfacing Servo Motor With Arduino : 6 Steps - Instructables, accessed on November 13, 2025, https://www.instructables.com/Interfacing-Servo-Motor-With-Arduino/
46. Servo Read Position - Interfacing - Arduino Forum, accessed on November 13, 2025, https://forum.arduino.cc/t/servo-read-position/18480
47. READ INTERNAL POTENTIOMETER IN A SERVO : r/arduino - Reddit, accessed on November 13, 2025, https://www.reddit.com/r/arduino/comments/hlrfib/read_internal_potentiometer_in_a_servo/
48. ~~Dissect~~ Parsing Commands - Programming - Arduino Forum, accessed on November 13, 2025, https://forum.arduino.cc/t/dissect-parsing-commands/1141601
49. Making servo stop at accurate values - Interfacing - Arduino Forum, accessed on November 13, 2025, https://forum.arduino.cc/t/making-servo-stop-at-accurate-values/17265
50. Servomotor Control with Limits and Variable Speed - DigiKey, accessed on November 13, 2025, https://www.digikey.com/en/blog/servomotor-control-with-limits-and-variable-speed
51. Helix: A Vision-Language-Action Model for Generalist Humanoid Control - Figure AI, accessed on November 13, 2025, https://www.figure.ai/news/helix

52. Embodying Language Models in Robot Action, accessed on November 13, 2025, https://www.esann.org/sites/default/files/proceedings/2024/ES2024-143.pdf
53. VLAS: Vision-Language-Action Model With Speech Instructions For Customized Robot Manipulation - Semantic Scholar, accessed on November 13, 2025, https://www.semanticscholar.org/paper/VLAS%3A-Vision-Language-Action-Model-With-Speech-For-Zhao-Ding/f6075ce4db8dcc359c3ac97b85f2644dc5cadda6
54. VLAS: Vision-Language-Action Model with Speech Instructions for Customized Robot Manipulation | OpenReview, accessed on November 13, 2025, https://openreview.net/forum?id=K4FAFNRpko
55. VLAS: Vision-Language-Action Model with Speech Instructions for Customized Robot Manipulation - arXiv, accessed on November 13, 2025, https://arxiv.org/html/2502.13508v2
56. [2502.13508] VLAS: Vision-Language-Action Model With Speech Instructions For Customized Robot Manipulation - arXiv, accessed on November 13, 2025, https://arxiv.org/abs/2502.13508
57. VLAS: Vision-Language-Action Model With Speech Instructions For Customized Robot Manipulation | alphaXiv, accessed on November 13, 2025, https://www.alphaxiv.org/overview/2502.13508v1
58. VLAS: Vision-Language-Action Model with Speech Instructions for Customized Robot Manipulation - arXiv, accessed on November 13, 2025, https://arxiv.org/html/2502.13508v1
59. Day 18: Hotword detection with Python - DEV Community, accessed on November 13, 2025, https://dev.to/picovoice/day-18-hotword-detection-with-python-5908
60. Personal AI Voice Assistant in Python with Vapi - YouTube, accessed on November 13, 2025, https://www.youtube.com/watch?v=91fv7QIcZcQ
61. Web calls - Vapi Docs, accessed on November 13, 2025, https://docs.vapi.ai/quickstart/web
62. Hot-word Detection and Voice Control| Part 2 of how to build a Virtual Assistant - YouTube, accessed on November 13, 2025, https://www.youtube.com/watch?v=BBGC25es5gk