A PROJECT REPORT

on

# Voice Braille Calculator

*Submitted by*


## Ms. Shravani Ganesh Wadkar


*in partial fulfillment for the award of the degree*

*of*

## BACHELOR OF SCIENCE

in

## COMPUTER SCIENCE

*under the guidance of*

## Prof.Tejal Wagh

**Department of Computer Science**



**Modern Education Society's**

**The D. G. Ruparel College of Arts, Science & Commerce**


**(Sem V)**

**(2025 – 2026)**


# Modern Education Society's
# The D. G. Ruparel College of Arts, Science & Commerce,

**senapati bapat marg, opp. Matunga road station (w.r.), mahim, mumbai 400 016**

## Department of Computer Science

## CERTIFICATE

This is to certify that Mr./Ms. Shravani Ganesh Wadkar of **T.Y.B.Sc. (Sem V)** class has satisfactorily completed the Project Voice Braille Calculator , to be submitted in the partial fulfillment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2025 – 2026**.

**Date of Submission:**

**Project Guide**

**Head / Incharge,**
**Department Computer Science**

**College Seal**

**Signature of Examiner**

# DECLARATION

I, Shravani Ganesh Wadkar, hereby declare that the project entitled "Voice Braille Calculator" submitted in the partial fulfillment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2025 – 2026** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

**Signature of the Student:**

**Place:**

**Date:**

# ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed directly or indirectly to the successful completion of my project titled **"Voice Braille Calculator** for Visually Impaired**"**. First and foremost, I extend my heartfelt thanks to my project guide Prof. Tejal Wagh their continuous guidance, encouragement, and valuable suggestions throughout the project. Their insights and support have been instrumental in shaping the direction and outcome of this work.

I sincerely thank my classmates, friends, and family members for their constant motivation, cooperation, and moral support. Their encouragement inspired me to put in my best efforts. Finally, I am thankful to all the open-source communities, online resources, and forums that provided the technical knowledge and tools which played a key role in the development of this project.

Without the guidance, support, and encouragement from all these people, this project would not have been possible.

# Table Of Content

# Table of Figures

# Table of Components

# D. G. Ruparel College of Arts, Science and Commerce

Department of Information Technology and Computer Science

2025-26

T.Y.B.Sc (Computer Science) - Project Synopsis (Sem-V)

**Name of the Student : Shravani Ganesh Wadkar**

**Roll No. :** CS-8144

**Title of the Project : Voice Braille Calculator**

**Name of the Project Guide : Ms. Tejal Wagh**

---

**Introduction:**

This project introduces an innovative system called the Advanced Voice + Braille Calculator for Visually Impaired. The system is designed to assist blind and visually impaired individuals in performing mathematical calculations independently. Traditional talking calculators only provide voice input/output, whereas this system integrates both voice and Braille input methods along with voice, LCD. It also supports advanced mathematical operations such as percentage, square root, modulus, and power. The application bridges the accessibility gap in education and daily usage by allowing users to input data either through speech recognition or Braille buttons, and receive results through multiple channels. This makes it more inclusive, versatile, and useful in real-life scenarios such as schools, colleges, and personal use by visually impaired individuals. This project not only promotes accessibility but also demonstrates how AI, embedded systems, and assistive technology can come together to enhance the lives of differently-abled individuals.

**Objective:**

The main objective of this project is to design and implement a calculator system that is fully accessible to visually impaired individuals. The project also focuses on error handling, user-friendliness, and future scalability

1. To design and develop a calculator system accessible to visually impaired users**.**
2. To provide dual input modes – Voice input (via microphone) and Braille input (via 6-button keypad).
3. To provide dual output modes – Voice output (via text-to-speech) and LCD display output.

4. To optionally include tactile Braille output using servo motors for enhanced accessibility.
5. To support basic and advanced mathematical operations – addition, subtraction, multiplication, division, percentage, power, and square root.
6. To implement error handling mechanisms (e.g., invalid input, no input, division by zero).
7. To ensure the system is modular, scalable, and extendable with features like and wireless connectivity in the future.
8. To create a low-cost, Arduino-based assistive device that is simple to build, maintain, and scale for educational institutions and NGOs supporting the visually impaired.
9. To promote inclusive education and technology, ensuring visually impaired users can independently solve mathematical problems just like sighted users.

**Scope:**

The scope of this project is to implement a fully functional calculator system that combines both voice and Braille input methods along with voice, LCD, and optional Braille servo output. In the current implementation, the system supports fundamental and advanced operations such as addition, subtraction, multiplication, division, percentage, power, and square root, with integrated error handling for invalid inputs.

The new system improves upon traditional talking calculators by introducing the following features:

**Dual Input Options:**

- Voice input using speech recognition.
- Braille input using a 6-button Braille keypad.

**Dual Output Options:**

- Voice output through text-to-speech.
- LCD display output for visible results.

**Optional Tactile Braille Output:**

- Servo motors to physically raise Braille pins for result digits.

**Advanced Mathematical Operations:**

- In addition to basic operations (+, –, ×, ÷), the system supports percentage, power, and square root.

**Error Handling:**

- Guides users with friendly messages when invalid input or division by zero occurs

**Methodology:**

The project follows a modular approach combining hardware and software integration to achieve accessibility for visually impaired users. The methodology involves the following steps:

**Step 1**: Requirement Analysis

Identify the needs of visually impaired users:

Voice input interface

Braille tactile feedback

Support for complex operations

Choose appropriate hardware and software tools based on accessibility and cost.

**Step 2**: Voice Recognition Module (Python)

Use the speech_recognition library to capture user voice commands through a microphone.

Convert natural language math phrases into symbolic expressions.

**Step 3**: Math Expression Processing

Sanitize and translate spoken inputs using regular expressions:

“plus” → “+”, “divided by” → “/”, “square root of” → math.sqrt()

Evaluate expressions using Python’s eval() or math library for advanced functions:

Percentage, power, mod, square root, etc.

**Step 4**: Serial Communication to Arduino

Convert the calculated result into a string (e.g., "5" or "23") and send it to Arduino via USB using pyserial.

Ensure reliable communication between the Python script and Arduino board.

**Step 5**: Braille Output Module (Arduino)

Receive the result on Arduino and split it into individual digits.

For each digit, activate servo motors/solenoids corresponding to the 6-dot Braille pattern.

The user can feel the result through raised pins arranged in a Braille format.

**Step 6**: Output Voice Feedback

Once the result is calculated, the system gives audio feedback to the user using text-to-speech (pyttsx3).

**Step 7**: Testing and Evaluation

Test the system for:

Accuracy of speech recognition

Correct math calculation

Accuracy of Braille dot patterns

**Step 8**: Optimization and Error Handling

Add exception handling in Python for invalid inputs or evaluation errors.

**Tools and Technologies:**

**Hardware Components:**

- Arduino Uno (Microcontroller)
- 8x2 LCD with I2C module (Display)
- Braille push buttons (Input)
- Push button (Voice trigger)
- Servo motors ( Braille tactile output)
- Breadboard
- Resistors
- jumper wires

**Software / Programming:**

**Front End:**

- LCD Display (output interface)
- Voice feedback via Python (pyttsx3)

**Back End:**

- Python (expression evaluation, speech recognition, text-to-speech)
- Arduino IDE (firmware for hardware interaction)

**Libraries / Frameworks:**

- speechrecognition (Voice input)
- pyttsx3 (Text-to-speech)
- pyserial (Python-Arduino communication)

**Platform:**

- PC/Laptop with Python installed
- Arduino IDE for code deployment

**Timeline:**

- **Analysis Phase: 12 Aug – 31 Aug**
  Understanding problem statement, identifying requirements of visually impaired users, and reviewing existing talking calculators.
- **Design Phase: 1 Sep – 10 Sep**
  Designing system architecture, Braille keypad layout, and Arduino-Python communication workflow.
- **Implementation Phase: 11 Sep – 20 Sep**
  Hardware setup (Arduino, LCD, buttons). Python coding for voice recognition, expression evaluation, and text-to-speech. Integration of Braille input and optional servo output.
- **Testing Phase: 20 Sep – 25 Sep**
  Functional testing of all operations (+, –, ×, ÷, %, sqrt, power).
  Error handling validation.

**Resources:**

**Hardware Resources:**

1. Arduino Uno microcontroller board
2. 8x2 LCD display with I2C module
3. 6 Braille push buttons + 1 trigger button

4. Breadboard, jumper wires, resistors (10kΩ)
5. USB cable for Arduino-PC connection
6. Servo motors (for Braille tactile output)
7. Microphone (for voice input)
8. Speaker or headphones (for audio output)

**Software Resources:**

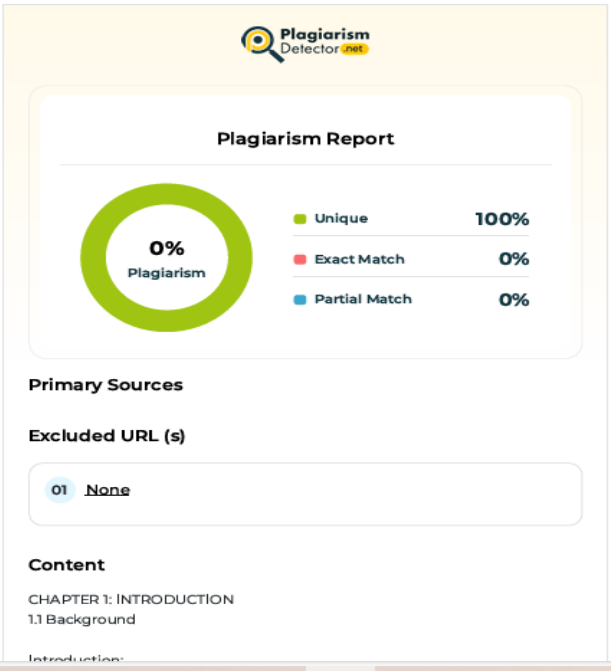
1. Arduino IDE (for programming Arduino)
2. Python (for expression evaluation and speech functions)
3. Required Python libraries: speechrecognition, pyttsx3, pyserial

**Expected Outcome:**

The final system will be an Advanced Voice + Braille Calculator that enables visually impaired users to perform mathematical calculations independently. The system integrates both voice and Braille input methods and provides results through voice, LCD display, and optional tactile Braille output**.**

- The user can choose voice mode, where a push button triggers Python to capture speech, recognize the spoken mathematical expression, evaluate it, and return the result through speech and LCD display.
- Alternatively, in Braille mode, the user enters input using a 6-button Braille keypad connected to Arduino, which transmits the expression to Python for evaluation.
- The system supports basic arithmetic operations (addition, subtraction, multiplication, division) and advanced operations (percentage, power, square root).
- Built-in error handling ensures the user is guided in cases of invalid input or undefined operations (like division by zero).
- An optional Braille servo output module allows the result to be physically represented in Braille dots using servo motors.

# PLAGIARISM REPORT



Plagiarism Detector .net

Plagiarism Report

0% Plagiarism

| | |
|---|---|
| Unique | 100% |
| Exact Match | 0% |
| Partial Match | 0% |

**Primary Sources**

**Excluded URL (s)**

01 None

**Content**

CHAPTER 1: INTRODUCTION
1.1 Background

Introduction:

CHAPTER 1: INTRODUCTION
1.1 Background

Introduction:
This project introduces an innovative system called the Advanced Voice + Braille Calculator for Visually Impaired. The system is designed to assist blind and visually impaired individuals in performing mathematical calculations independently. Traditional talking calculators only provide voice input/output, whereas this system integrates both voice and Braille input methods along with voice, LCD. It also supports advanced mathematical operations such as percentage, square root, modulus, and power. The application bridges the accessibility gap in education and daily usage by allowing users to input data either through speech recognition or Braille buttons, and receive results through multiple channels. This makes it more inclusive, versatile, and useful in real-life scenarios such as schools, colleges, and personal use by visually impaired individuals. This project not only promotes accessibility but also demonstrates how AI, embedded systems, and assistive technology can come together to enhance the lives of differently-abled individuals.

Evaluation:
The Voice + Braille Calculator successfully demonstrates an accessible, low-cost assistive technology.
It combines Arduino hardware (buttons + LCD) with Python software (voice recognition + speech output).
While currently limited to basic math, the project can be enhanced into a full-fledged learning tool for visually impaired students.

# CHAPTER 1: INTRODUCTION

## 1.1 Background

## Introduction:

This project introduces an innovative system called the Advanced Voice + Braille Calculator for Visually Impaired. The system is designed to assist blind and visually impaired individuals in performing mathematical calculations independently. Traditional talking calculators only provide voice input/output, whereas this system integrates both voice and Braille input methods along with voice, LCD. It also supports advanced mathematical operations such as percentage, square root, modulus, and power. The application bridges the accessibility gap in education and daily usage by allowing users to input data either through speech recognition or Braille buttons, and receive results through multiple channels. This makes it more inclusive, versatile, and useful in real-life scenarios such as schools, colleges, and personal use by visually impaired individuals. This project not only promotes accessibility but also demonstrates how AI, embedded systems, and assistive technology can come together to enhance the lives of differently-abled individuals.

## Evaluation:

The Voice Braille Calculator successfully demonstrates an accessible, low-cost assistive technology**.**
It combines Arduino hardware (buttons + LCD) with Python software (voice recognition + speech output).
While currently limited to basic math, the project can be enhanced into a full-fledged learning tool for visually impaired students.

## 1.2 Objectives

The main objective of this project is to design and implement a calculator system that is fully accessible to visually impaired individuals. The project also focuses on error handling, user-friendliness, and future scalability

**1. To design and develop a calculator system accessible to visually impaired users.**

The main purpose of this project is to design and develop a **user-friendly calculator** that caters specifically to visually impaired individuals, who face difficulties using traditional calculators with visual screens and small buttons. This system removes the dependency on sight by enabling audio and tactile interactions through voice input and Braille-based tactile buttons.
It ensures that visually impaired users can perform mathematical calculations independently, enhancing their confidence and inclusion in educational and professional environments.

**2. To provide dual input modes – Voice input (via microphone) and Braille input**

The system supports two modes of input:

> **Voice Input**: The user can speak mathematical expressions such as "two plus three" or "square root of eighty-one," which the system interprets using speech recognition technology in Python (speech_recognition library).
>
> **Braille Input**: For users familiar with Braille, a button is provided, allowing entry of digits and operations using combinations similar to Braille dot patterns.
> This dual-mode input design ensures accessibility for users with varying comfort levels whether they prefer speaking or tactile interaction.

**3. To provide dual output modes – Voice output (via text-to-speech) and LCD display output.**

The system provides two forms of output:

- **Voice Output**: The final result is spoken aloud using Python's pyttsx3 text-to-speech engine, helping visually impaired users hear their calculation results.
- **LCD Display Output**: The result is also displayed on a 16x2 LCD screen connected to the Arduino, enabling sighted users (teachers, assistants, or evaluators) to see the result simultaneously.
  This ensures collaborative usability, where both the user and observer can interact with the device in real time.

4. **To support basic and advanced mathematical operations – addition, subtraction, multiplication, division, percentage, power, and square root.**

The calculator is not limited to simple arithmetic. It supports a wide range of mathematical operations, including:

- **Basic**: Addition (+), Subtraction (−), Multiplication (×), Division (÷)
- **Advanced**: Percentage (%), Power (^), Square Root (√)
  The voice recognition module intelligently converts spoken phrases such as "two to the power of three" or "square root of eighty-one" into mathematical expressions that are processed and evaluated by the Python program.

**5. To implement error handling mechanisms (e.g., invalid input, no input, division by zero).**

Robust error handling is implemented to ensure smooth operation and user safety:

- If no input is detected, the system prompts the user to "Please repeat."
- If an invalid expression or unknown word is spoken, the system responds with "Error in calculation."
- Special checks prevent division by zero or unsupported operations.
  These mechanisms prevent the program from crashing and guide the user to reattempt their calculation — crucial for accessibility and ease of use.

**6. To ensure the system is modular, scalable, and extendable with features like Bluetooth and wireless connectivity in the future.**

The project's architecture is designed in a modular structure, with separate modules for:

- Voice processing
- Braille input handling
- Arduino communication
- Expression evaluation
- Output delivery

**7. To create a low-cost, Arduino-based assistive device that is simple to build, maintain, and scale for educational institutions and NGOs supporting the visually impaired.**

The project uses readily available, low-cost components such as an Arduino Uno**,** 16x2 LCD**,** push buttons**,** microphone**,** and speaker**.**
Its affordability and simplicity make it an excellent solution for schools for the blind, NGOs, and

rural education centers, where budget constraints exist.

**8. To promote inclusive education and technology, ensuring visually impaired users can independently solve mathematical problems just like sighted users**.

The ultimate goal of this project is to promote inclusive education by bridging the technological gap between sighted and visually impaired learners.
By allowing users to perform mathematical operations through voice and tactile feedback, the system supports the principle of **"**Technology for All**."**
It empowers visually impaired students to work independently, improving their learning experience, confidence, and participation in mainstream education.

## 1.3 Purpose, Scope and Applicability

### 1.3.1 Purpose:

**1. To provide a calculator that blind and visually impaired users can use independently**.

The core aim of the project is to develop a calculator that empowers blind and visually impaired users to perform mathematical operations without assistance**.**
Traditional calculators depend heavily on visual displays and small buttons, making them unusable for visually challenged individuals.
This system replaces the need for visual feedback by introducing audio-based interaction and tactile input, allowing users to operate the calculator confidently and independently.
It helps users perform academic and everyday calculations without requiring help from sighted individuals, thus promoting self-reliance and accessibility.

**2. To combine voice input/output with Braille-style button input for flexibility**.

The project integrates dual input and output methods to cater to different user preferences and abilities:

- **Voice Input & Output**: Users can speak calculations like "five multiplied by two" and hear the result through a clear synthesized voice.
- **Braille-style Buttons**: Users familiar with Braille can enter numbers and operations using tactile buttons connected to the Arduino.
  This dual-mode approach provides flexibility — users can switch between voice or Braille input depending on their comfort or environment (e.g., noisy surroundings).
  The combination ensures the device is universally accessible across a wide range of users with different levels of vision and experience.

**3. To make mathematics more accessible in education and daily life.**

Mathematics is a crucial skill in both education and daily activities such as shopping, banking, and measurements.Visually impaired students often rely on assistance for simple mathematical tasks due to the lack of accessible calculators.This project bridges that gap by offering an inclusive learning tool that supports audio feedback and tactile interaction.
By integrating this system in schools, NGOs, and homes, visually impaired individuals can learn, practice, and apply mathematics independently, fostering inclusive education and equal participation in classrooms and society.

**4. To build an affordable, portable assistive technology using Arduino and Python**.

Affordability and portability are key considerations in this project.
The calculator is developed using Arduino Uno**,** 16x2 LCD**,** microphone, and **s**peaker all low-cost and easily available components.Python is used for voice processing and mathematical evaluation, offering flexibility and high accuracy without expensive hardware.The system is lightweight, battery-compatible, and cost-effective, making it ideal for educational institutions, rural schools, and NGOs that support the visually impaired.The design is simple, easy to reproduce, and scalable ensuring that it can be adapted for larger implementations or commercial production in the future.

## 1.3.2 Scope:

The scope of this project is to implement a fully functional calculator system that combines both voice and Braille input methods along with voice, LCD, and optional Braille servo output. In the current implementation, the system supports fundamental and advanced operations such as addition, subtraction, multiplication, division, percentage, power, and square root, with integrated error handling for invalid inputs.

The new system improves upon traditional talking calculators by introducing the following features:

**Dual Input Options:**

- Voice input using speech recognition.
- Braille input using a 6-button Braille keypad.

**Dual Output Options:**

- Voice output through text-to-speech.

- LCD display output for visible results.

**Advanced Mathematical Operations:**

- In addition to basic operations (+, –, ×, ÷), the system supports percentage, power, and square root.

**Error Handling:**

- Guides users with friendly messages when invalid input or division by zero occurs.

## 1.3.3 Applicability:

**1. Blind/Visually Impaired Students – Helps in learning and performing mathematics independently**.

This project primarily benefits blind and visually impaired students who face difficulties using standard calculators.
With the inclusion of voice input/output and Braille-style tactile buttons, the calculator enables them to perform mathematical operations independently, without requiring a sighted person's help.It encourages self-learning, boosts confidence, and promotes equal participation in academic environments.The system thus becomes an important assistive learning tool in special schools and inclusive classrooms for the visually impaired.

**2. Inclusive Classrooms – Teachers can see results on LCD while students hear them via voice.**

In an inclusive classroom setup, where both visually impaired and sighted students learn together, this calculator plays a key bridging role.
While visually impaired students receive audio feedback for their calculations, teachers or peers can simultaneously view the same result on the LCD display connected to the Arduino.
This dual-output design ensures better communication, monitoring, and collaboration between teachers and students.It aligns perfectly with the principles of inclusive education allowing visually impaired learners to use the same tools as their sighted peers.

**3. Examinations / Educational Tools – Provides a fair way for visually impaired students to solve problems**.

During examinations, visually impaired students often rely on scribes or oral assistance to perform calculations.
This device can serve as a fair, independent examination aid approved by educational boards, as it removes human dependency while maintaining accessibility.
It ensures that visually impaired students can attempt mathematical questions independently using the same logic and operations as sighted students.
Additionally, the system can be used in practical labs, training sessions, and math learning modules as an educational support tool.

**4. Daily Life – Can be used for simple arithmetic in shops, banking, and personal finance by blind users.**

Beyond academic use, the calculator is practical for everyday applications in daily life.
Visually impaired users can use it for simple arithmetic operations such as totaling bills, calculating discounts, dividing expenses, or managing personal finances.
With its portable, speech-enabled interface, the system acts as a daily-use companion that improves financial independence and convenience for visually impaired individuals in real-world environments like shops or banks.

**5. Rural / Low-Power Areas – With solar/battery add-on, it can be used where electricity is unreliable.**

The project's design can be enhanced with a solar or battery power module, making it fully functional even in rural or low-power regions.
This feature ensures that visually impaired users in remote areas can benefit from the calculator without depending on continuous electrical power.
It aligns with the goal of creating sustainable and accessible technology for all — particularly for resource-limited schools or NGOs that work in underprivileged regions.

## 1.4 Achievement

- **Accessible Calculator Built**
    - Designed and implemented a calculator that visually impaired users can operate independently.
- **Dual Input System**
    - Integrated hardware button input (Braille-style) and voice input (speech recognition).
- **Dual Output System**
    - Provided voice feedback through Python text-to-speech and visual output on 16x2 LCD.
- **Error Handling**
    - Implemented safeguards against invalid inputs, division by zero, and incorrect syntax.
- **Support for Advanced Operations**
    - Added percentage, power, and square root calculations in addition to basic arithmetic.
- **Arduino–Python Integration**
    - Successfully established serial communication between Arduino and Python for data exchange.
- **User-Friendly Design**
    - Simple button layout with Braille marking for digits and operators.
    - Clear, step-by-step voice guidance for results.
- **Low-Cost Solution**
    - Built using affordable and widely available components (Arduino Uno, LCD, push buttons, microphone, speaker).

## 1.5 Gantt Chart

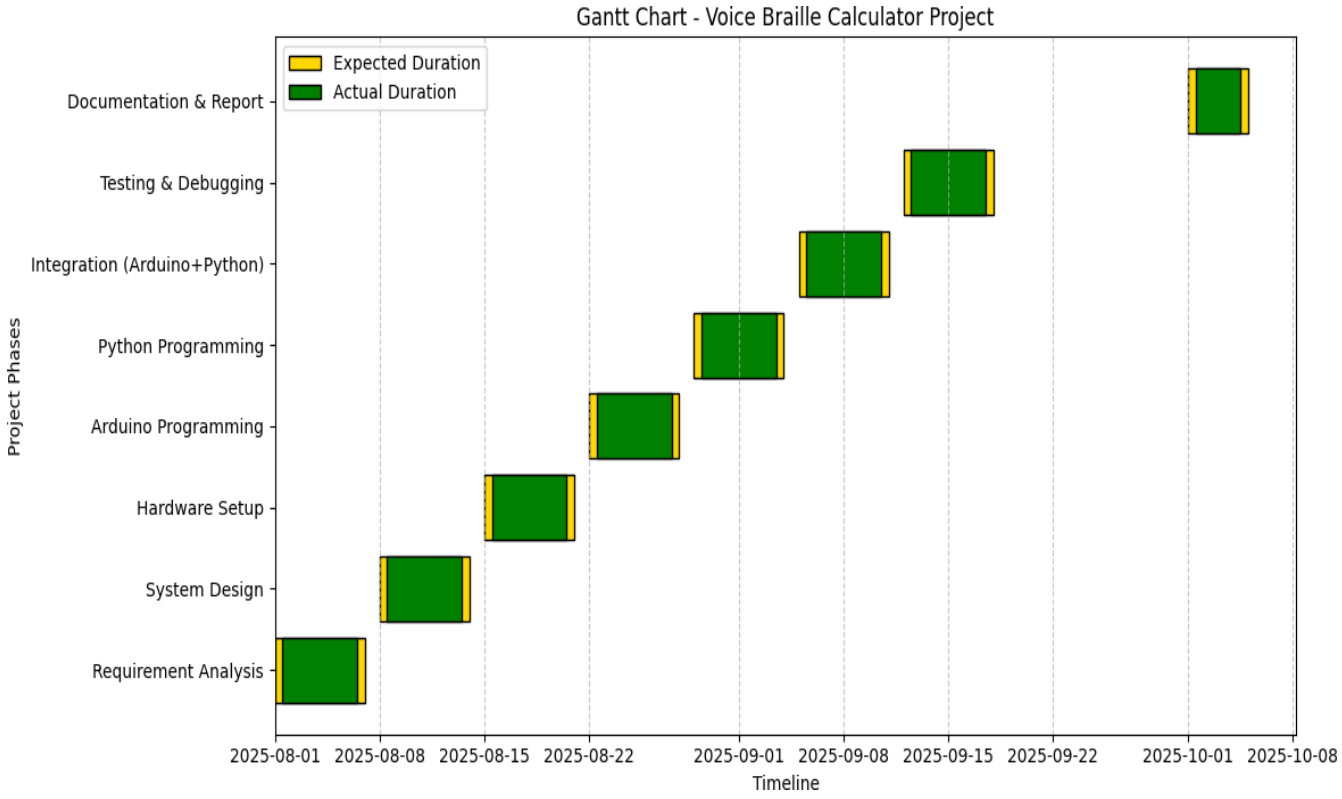



Figure 1: Gantt Chart

# CHAPTER 2: REQUIREMENT AND ANALYSIS

## 2.1 Problem Definition

Visually impaired individuals face significant challenges when performing mathematical operations using traditional calculators or computing devices.
Conventional calculators are purely visual tools, designed with small numeric buttons and screen-based outputs that depend entirely on sight.
As a result, blind and low-vision users cannot independently operate these devices, relying instead on sighted assistance or specialized, expensive equipment.

While some accessible calculators exist, they often suffer from one or more of the following issues:

1. **High Cost:** Commercial talking or Braille-based calculators are expensive and unaffordable for most students or educational institutions, particularly in developing regions.
2. **Limited Availability:** Such devices are rarely available in rural or under-resourced schools where accessibility tools are most needed.
3. **Lack of Dual Modes:** Most devices support either Braille or voice but not both, restricting flexibility for users with different preferences or abilities.
4. **Poor Integration with Education:** Existing systems are not easily usable in inclusive classrooms, where teachers need to view results and students need audio feedback simultaneously.
5. **Dependency on Constant Power:** Many devices require continuous electrical power and cannot operate in areas with poor electricity availability.

Because of these limitations, visually impaired students struggle to perform even basic arithmetic or participate fully in mathematics education. This dependency negatively affects their academic performance, confidence, and independence, making it difficult for them to engage equally in classrooms or everyday financial tasks.

## 2.2 Requirement Specification

### 1. Functional Requirements

These define what the system must do:

1. The system shall allow numeric input (0–9) through hardware buttons.
2. The system shall allow arithmetic operations **(+, −, ×, ÷)** through hardware buttons.
3. The system shall allow special operations like power **(^)** and square root **(**sqrt**)**.
4. The system shall process complete expressions only when the Enter (=) button is pressed.
5. The system shall support voice mode, where the user can speak a calculation (e.g., *"*square root of 81*"*).
6. The system shall provide dual output:
    - Voice output (Python text-to-speech).
    - LCD display output.
7. The system shall handle errors gracefully (e.g., invalid input, division by zero).
8. The system shall allow communication between Arduino and Python via Serial.

### 2. Non-Functional Requirements

These define the quality of the system:

1. **Usability** – Buttons should be arranged in a Braille-marked layout for easy identification by touch.
2. **Reliability** – System must consistently provide accurate results.
3. **Performance** – Each input should be processed and responded to within 1–2 seconds.

4. **Portability** – The system should be compact and optionally run on battery/solar power.
5. **Scalability** – System should allow future extensions (scientific functions, multilingual voice, Braille servo output).
6. **Affordability** – System must use low-cost, easily available components.

## 2.3 Planning and Scheduling

# 1. Project Planning

The project is planned in **phases**, starting from requirement gathering to final demonstration:

1. **Requirement Analysis**
    - Identify needs of visually impaired users.
    - Decide features: dual input/output, basic + advanced operations.
2. **System Design**
    - Draw circuit diagram (Arduino + LCD + Buttons).
    - Design workflow for Arduino–Python communication.
3. **Hardware Implementation**
    - Connect push buttons, LCD, and Arduino.
    - Ensure stable power supply.
4. **Software Implementation**
    - Write Arduino code for input + LCD output.
    - Write Python code for expression evaluation, voice input, and TTS output.
5. **Integration**
    - Establish serial communication between Arduino and Python.
    - Test both voice mode and button mode together.
6. **Testing & Debugging**
    - Test all operations (+, −, ×, ÷, sqrt, power).
    - Check error handling and voice response accuracy.

7. **Documentation**
    - Prepare synopsis, report, diagrams, and presentation.
8. **Final Demonstration**
    - Showcase to external examiner with working hardware.

## 2. Scheduling:

| Phase | Duration | Dates |
|---|---|---|
| Requirement Analysis | 1 week | 1 Aug – 7 Aug |
| System Design | 1 week | 8 Aug – 14 Aug |
| Hardware Setup | 1 week | 15 Aug – 21 Aug |
| Arduino Programming | 1 week | 22 Aug – 28 Aug |
| Python Programming | 1 week | 29 Aug – 4 Sep |
| Integration (Arduino+Python) | 1 week | 5 Sep – 11 Sep |
| Testing & Debugging | 1 week | 12 Sep – 18 Sep |
| Documentation & Report | 5 days | 1 Oct – 5 Oct |

## 2.4 Software and Hardware Specification

### Hardware Specification

- **Microcontroller**: Arduino Uno

  Acts as the central control unit for hardware — reads button inputs (Braille), communicates with Python via serial, and drives the LCD display.

Figure1:Arduino Board

- **Display**: 16x2 LCD with I2C module

  Displays text results or messages from Python, allowing sighted users (teachers, assistants) to view the output.

Figure2:LCD

- **Input Devices**:

  Used for Braille-style input — digits (0–9) and operators (+, −, ×, ÷, =).

  - 10 Push buttons for digits (0–9)
  - 5 Push buttons for operators (+, −, ×, ÷, =)

Figure3:Push Button

- **Breadboard + Jumper Wires** (for prototyping)

  Enables flexible circuit connections between Arduino, buttons, and LCD.

Figure 4:Jumper Wire

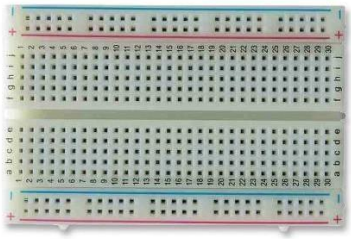


Figure5:Breadboard

- **Power Supply**: USB (5V from laptop)

  Provides power and serial communication between Arduino and PC.
- **Laptop/PC**: for running Python (voice input/output processing)
- **Microphone**: for capturing voice input
- **Speaker/Headphones**: for voice output to the user

## Software Specification

- **Operating System**: Windows 10
- **Arduino IDE**: for writing and uploading Arduino code
- **Python Version**: Python 3.12

## Required Python Libraries

- pyserial → For communication with Arduino
- pyttsx3 → For Text-to-Speech (voice output)
- speechrecognition → For capturing and processing voice input
- math → For mathematical functions (sqrt, power, etc.)

Arduino Libraries

- Wire.h → For I2C communication
- LiquidCrystal_I2C.h → For LCD display

# CHAPTER 3: SYSTEM DESIGN

## 3.1 Basic Module

1. Input Module

- **Button Input**:
    - 0–9 digits, +, −, ×, ÷, = operators.
    - Enter button finalizes the expression.
- **Voice Input** :
    - User speaks an expression (e.g., *"*square root of eighty one*"*).
    - Python's speech recognition converts it into text.

2. Processing Module

- **Arduino Processing**:
    - Reads button presses.
    - Sends expression to Python when "=" is pressed.
- **Python Processing**:
    - Receives expression from Arduino.
    - Evaluates using eval() and math library.
    - Handles special functions: sqrt, power (^).
    - Sends result back to Arduino.

3. Output Module

- **LCD Display**:

    - Shows entered expression.
    - Displays final result after evaluation.
- **Voice Output**:
    - Python uses pyttsx3 to speak the result.

4. Communication Module

- Serial Communication (USB) between Arduino and Python.
- Arduino → sends button inputs / expressions.
- Python → sends evaluated result back.

5. Error Handling Module

- Detects invalid input, wrong expressions, or division by zero.
- Shows “Error” on LCD.
- Speaks *“*Error in calculation*”* via Python.

## 3.2 Data Design

### 3.2.1 Schema Design

The system is divided into data flow layers:

1. **Input Layer (Data Capture)**
    - **Button Input** → Digit/operator codes (0–9, +, −, ×, ÷, =)
    - **Voice Input** → Spoken expression converted to text
2. **Processing Layer (Data Conversion & Evaluation)**
    - Arduino collects button inputs → builds expression string → sends to Python.
    - Python receives string → parses & evaluates using math library.
3. **Output Layer (Result Delivery)**
    - LCD Display → Shows expression and final result.
    - Voice Output → Announces final result.

| Source (Input) | Data Type | Processing | Destination (Output) |
|---|---|---|---|
| Button Press | Character (0–9, +, −, ×, ÷, =) | Arduino builds expression | Sent to Python |
| Voice Input | Speech → Text | Python converts text → math expression | Python evaluator |
| Expression | String | Python evaluates | LCD + Speech Output |
| Result | Numeric/String | Python → Arduino | LCD + Voice output |

### 3.2.2 Data Integrity and Constraint:

## Integrity:

Ensures correctness and consistency of data during input, processing, and output.

1. **Input Validation**
    - Only valid digits/operators are accepted from buttons.
    - Voice input is verified (e.g., “square root of eighty one” → sqrt(81)).
2. **Expression Integrity**
    - Arduino concatenates button inputs correctly in sequence.
    - Expression sent to Python only after pressing “=”.
3. **Result Integrity**
    - Python evaluates only safe mathematical expressions (`+`, `-`, `*`, `/`, `sqrt`, `^`).
    - Invalid expressions return “Error” instead of crashing.

## Constraint

1. **Operational Constraints**
    - Limited to basic operations (+, −, ×, ÷, sqrt, power).
    - Expression length limited by LCD (16 characters).
2. **Hardware Constraints**
    - Uses only 16x2 LCD → restricted display area.
    - Limited number of GPIO pins on Arduino → max number of buttons.
    - Requires USB connection to PC for Python-based voice processing.
3. **Software Constraints**

- Speech recognition requires a microphone and good clarity.
- eval() restricted to prevent unsafe code execution.
- Python libraries (speechrecognition, pyttsx3, pyserial) must be installed.

## 3.3 Logic Diagram

### 3.2.1 Event Table

| Event | Trigger (Cause) | Source | System Action | Output |
|---|---|---|---|---|
| **Digit Pressed** | User presses digit button (0–9) | Push Button → Arduino | Append digit to expression string | Show digit on LCD |
| **Operator Pressed** | User presses +, −, ×, ÷ button | Push Button → Arduino | Append operator to expression string | Show operator on LCD |
| **Enter Pressed** | User presses "=" button | Push Button → Arduino | Send complete expression to Python | LCD shows "Solving..." |
| **Expression Evaluation** | Arduino sends expression via Serial | Arduino | Python parses & evaluates expression | Sends result back to Arduino |
| **Result Displayed** | Python returns result | Python → Arduino | Arduino receives result | LCD shows result, Python speaks result |
| **Invalid Expression** | Wrong input (e.g., "2++3") | Arduino/Python | Error detected in processing | LCD shows "Error", Python speaks "Error in calculation" |
| **Division by Zero** | Expression contains /0 | Python evaluator | Exception raised | LCD shows "Error", Python speaks error |
| **Voice Mode Activated** | User presses Voice button | Push Button → Arduino | Arduino sends "VOICE" signal | Python activates microphone listening |
| **Voice Input Given** | User speaks expression | Microphone → Python | Speech recognition converts to text → expression | LCD shows expression, Python evaluates |
| **Voice Result Output** | Voice expression solved | Python | Python evaluates & speaks result | LCD shows result, Python announces result |

## 3.3.2 Use Case Diagram

A Use Case Diagram in Unified Modeling Language (UML) is a visual representation that illustrates the interactions between users (actors) and a system. It captures the functional requirements of a system, showing how different users engage with various use cases, or specific functionalities, within the system. Use case diagrams provide a high-level overview of a system's behavior, making them useful for stakeholders, developers, and analysts to understand how a system is intended to operate from the user's perspective, and how different processes relate to one another. They are crucial for defining system scope and requirements.

1. Actors

Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modeling system behavior.

2. Use Cases

Use cases are like scenes in the play. They represent specific things your system can do.

3. System Boundary

The system boundary is a visual representation of the scope or limits of the system you are modeling. It defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it. The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system.

4. Association Relationship

The Association Relationship represents a communication or interaction between an actor and a use case. It is depicted by a line connecting the actor to the use case. This relationship signifies that the actor is involved in the functionality described by the use case.

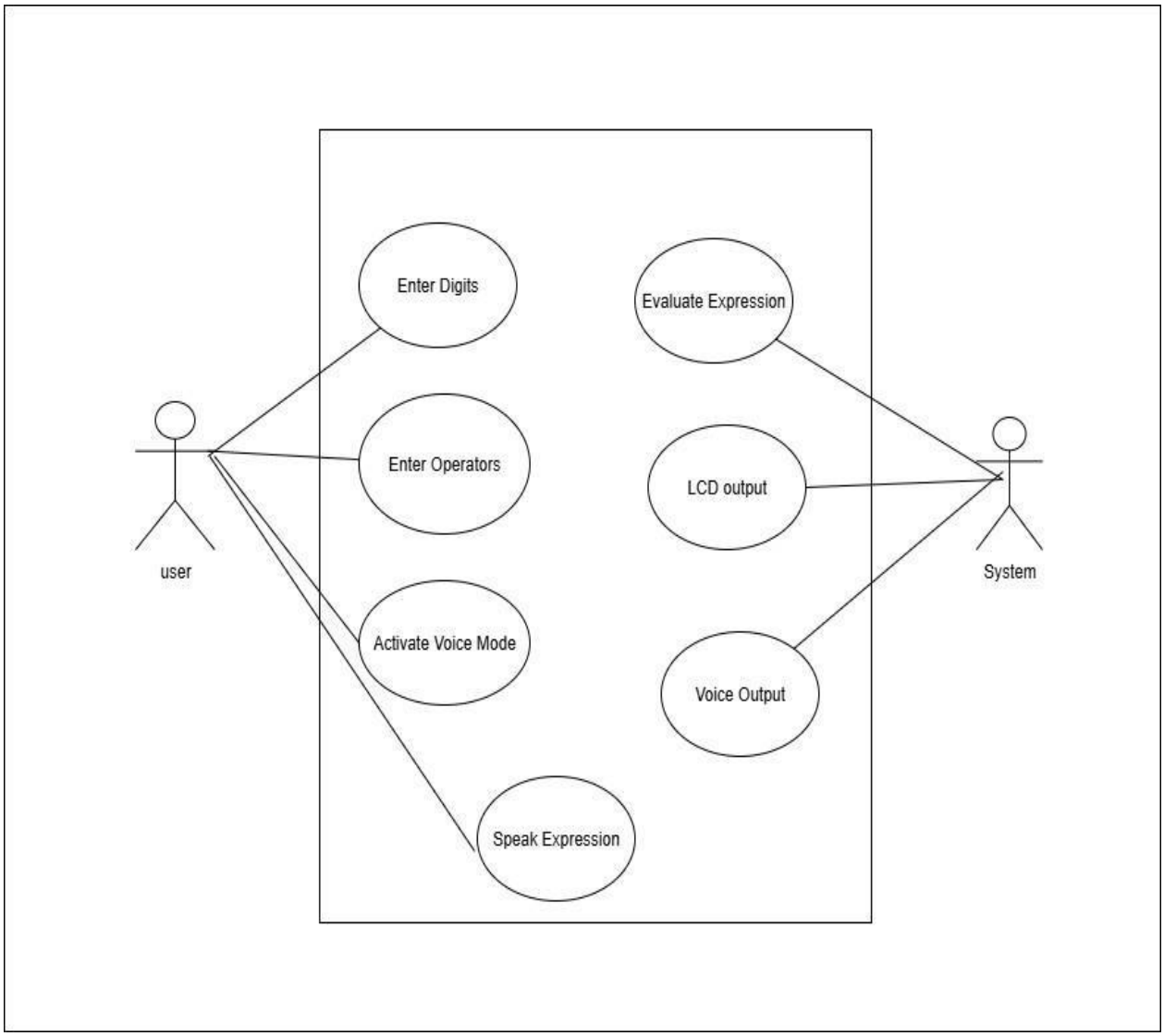



Figure 2: Use Case Diagram

### 3.3.3 Activity Diagram

Activity diagrams are an essential part of the Unified Modeling Language (UML) that help visualize workflows, processes, or activities within a system. They depict how different actions are connected and how a system moves from one state to another. By offering a clear picture of both simple and complex workflows, activity diagrams make it easier for developers and stakeholders to understand how various elements interact in a system.

1. Initial State
The starting state before an activity takes place is depicted using the initial state.

2. Action or Activity State
An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

3. Action Flow or Control flows
Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state.

4. Decision node and Branching
When we need to make a decision before deciding the flow of control, we use the decision node. The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.

5. Final State or End State
The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram. A system or a process can have multiple final states.

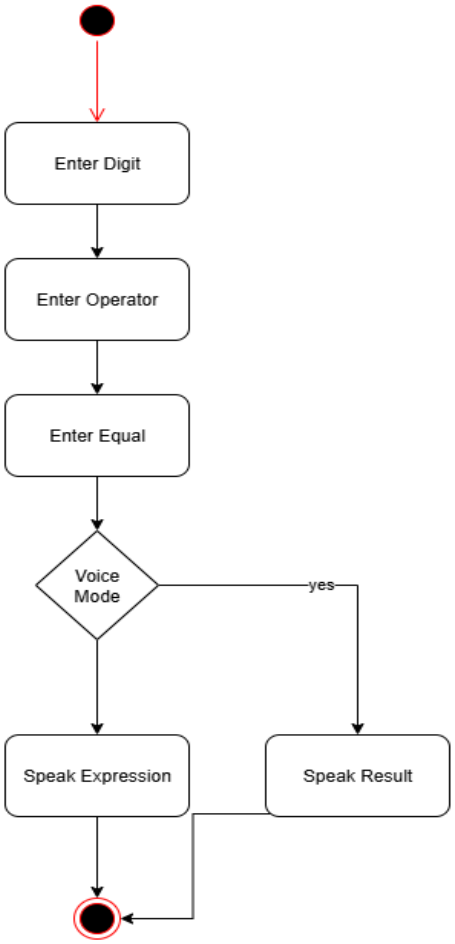



Figure 3: Activity Diagram

### 3.3.4 Class Diagram

A UML class diagram visually represents the structure of a system by showing its classes, attributes, methods, and the relationships between them.

1. **Class Name:**
   - The name of the class is typically written in the top compartment of the class box and is centered and bold.

2. **Attributes:**
   - Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.

3. **Methods:**
   - Methods, also known as functions or operations, represent the behavior or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.

4. **Visibility Notation:**
   - Visibility notations indicate the access level of attributes and methods. Common visibility notations include:
     - + for public (visible to all classes)
     - - for private (visible only within the class)
     - # for protected (visible to subclasses)
     - ~ for package or default visibility (visible to classes in the same package)

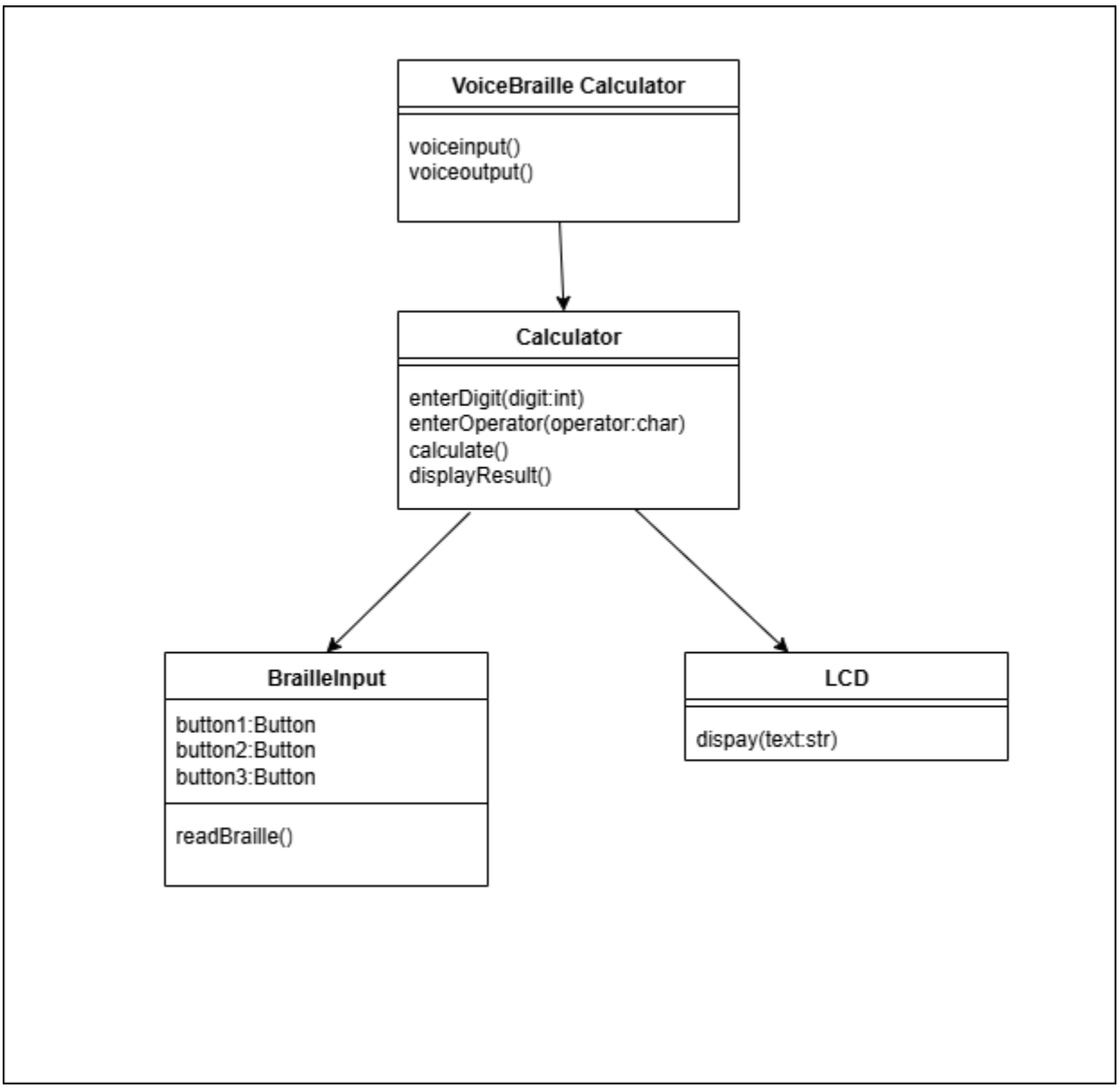



Figure 4: Class Diagram

## 3.3.5 Object Diagram

Object diagrams are a visual representation in UML (Unified Modeling Language) that illustrates the instances of classes and their relationships within a system at a specific point in time. They display objects, their attributes, and the links between them, providing a snapshot of the system's structure during execution. Since object diagrams depict behavior when objects have been instantiated, we can study the behavior of the system at a particular instant.

1. Objects or Instance specifications
When we instantiate a classifier in a system, the object we create represents an entity which exists in the system. We can represent the changes in object over time by creating multiple instance specifications. We use a rectangle to represent an object in an object diagram.

2. Attributes and Values
Inside the object box, attributes of the object are listed along with their specific values.

3. Link
We use a link to represent a relationship between two objects. We represent the number of participants on the link for each, at the end of the link. The term link is used to specify a relationship between two instance specifications or objects. We use a solid line to represent a link between two objects.

4. Dependency Relationships
We use a dependency relationship to show when one element depends on another element. A dependency is used to depict the relationship between dependent and independent entities in the system.

- Any change in the definition or structure of one element may cause changes to the other.
- This is a unidirectional kind of relationship between two objects.
- Dependency relationships are of various types specified with keywords like Abstraction, Binding, Realization, Substitution and Usage are the types of dependency relationships used in UML.

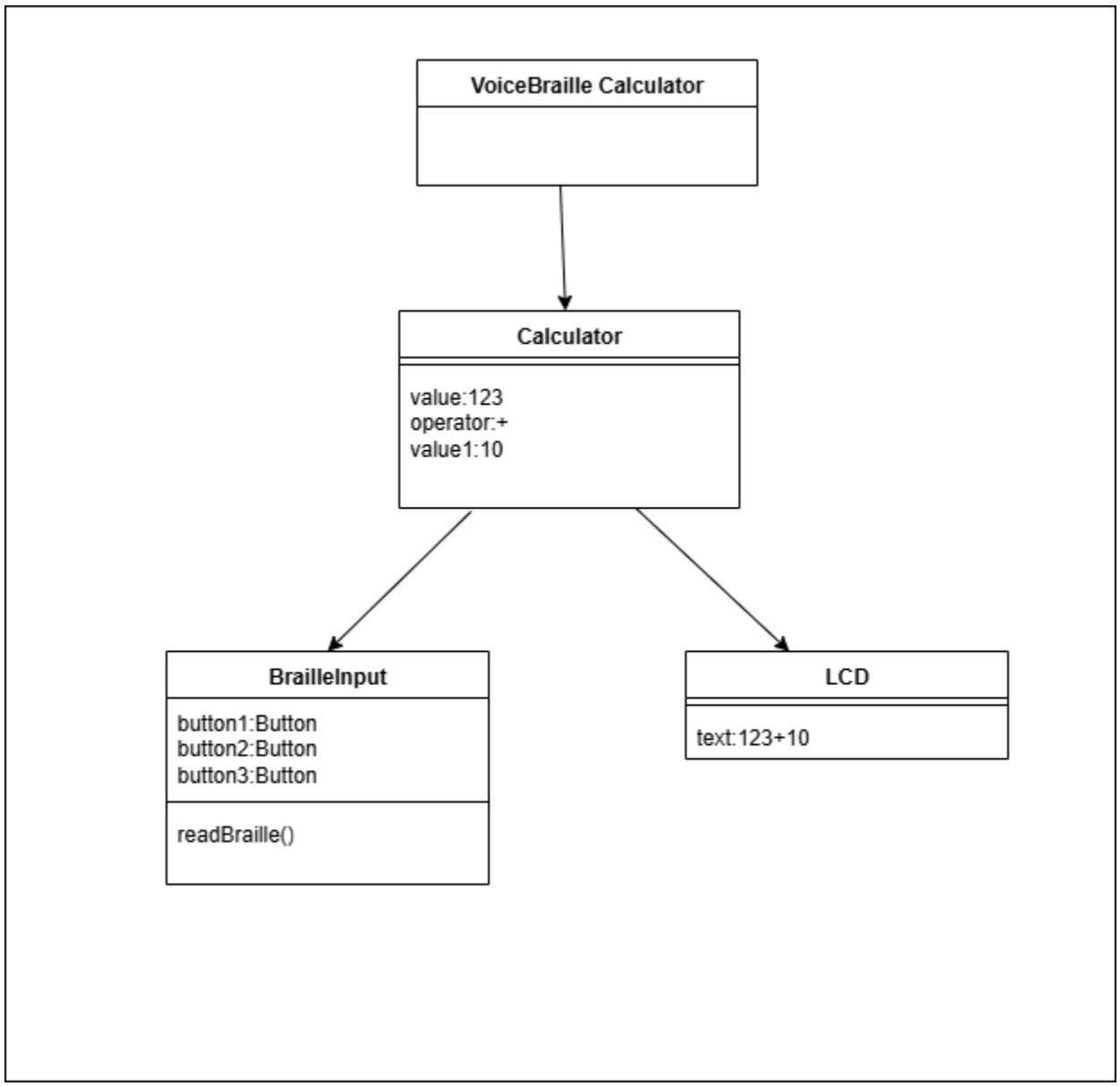



Figure 5: Object Diagram

## 3.3.6 Sequence Diagram

A Sequence Diagram is a key component of Unified Modeling Language (UML) used to visualize the interaction between objects in a sequential order. It focuses on how objects communicate with each other over time, making it an essential tool for modeling dynamic behavior in a system. Sequence diagrams illustrate object interactions, message flows, and the sequence of operations, making them valuable for understanding use cases, designing system architecture, and documenting complex processes.

**1. Actors**

An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

**2. Lifelines**

A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

**3. Messages**

Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline.

- We represent messages using arrows.
- Lifelines and messages form the core of a sequence diagram.

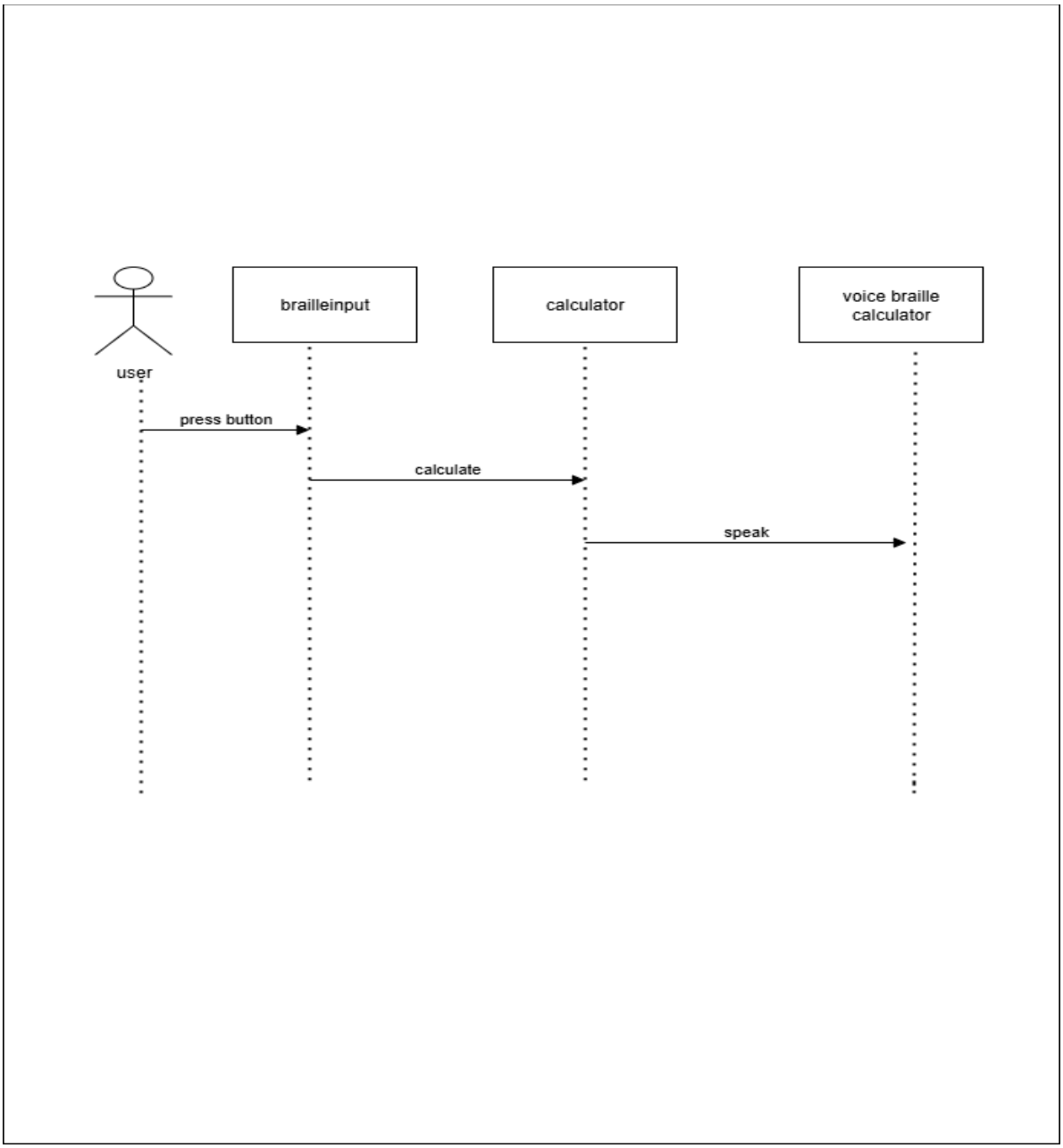



Figure 6: Sequence Diagram

## 3.3.7 State Chart Diagram

A State diagram is a UML diagram which is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions.

- State Machine diagrams are also known as State Diagrams and State-Chart Diagrams. These both terms can be used interchangeably.
- A state machine diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli (events that cause the system to change its state from one to another).
- We can say that every class has a state but we don't model every class using State Machine diagrams.

1. Initial state

We use a black filled circle represent the initial state of a System or a Class.

2. Transition

We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

3. State

We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

4. Final State

We use a filled circle within a circle notation to represent the final state in a state machine diagram.

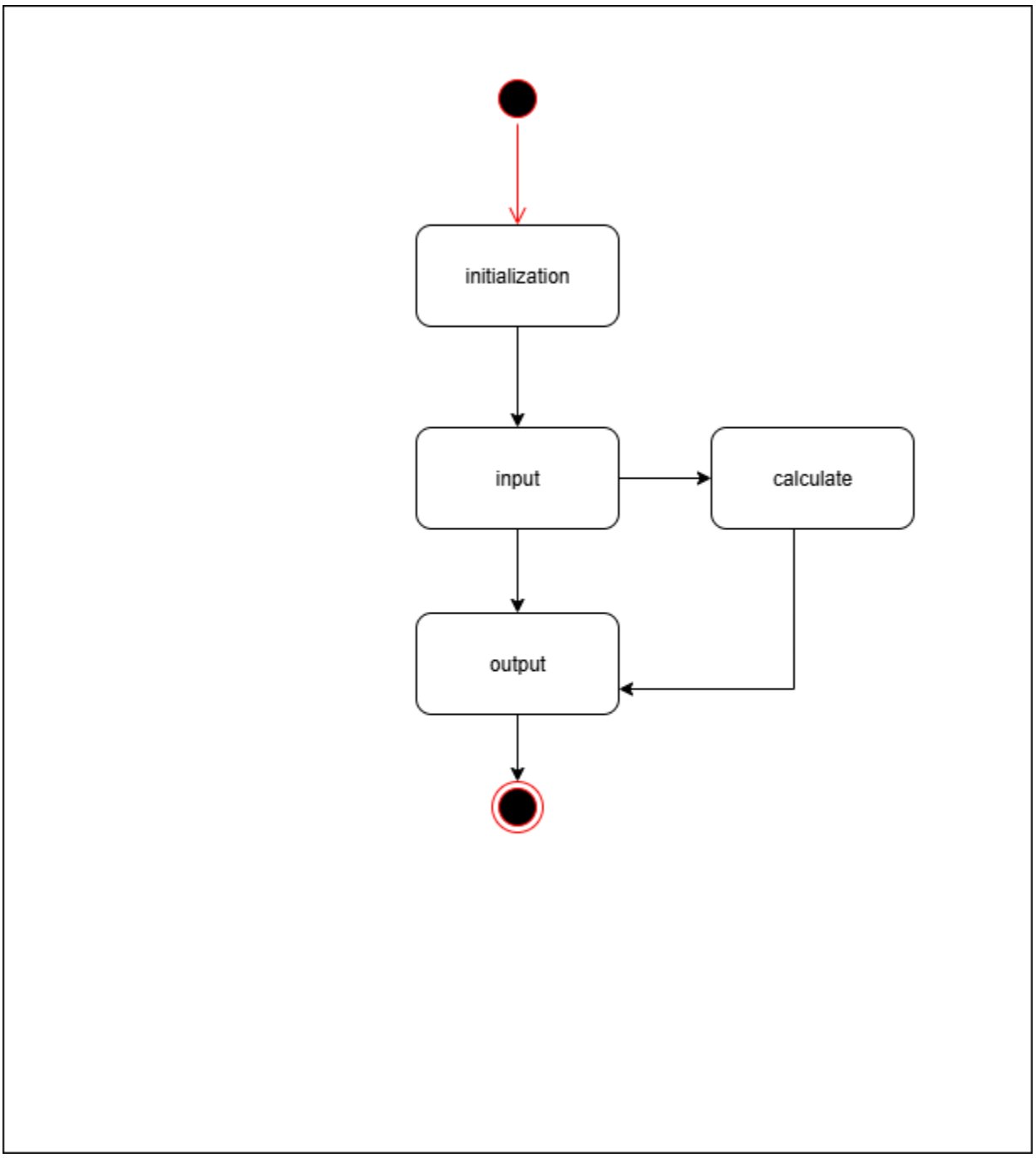



Figure 7: State Diagram

## 3.3.8 Component Diagram:

Component-based diagrams are essential tools in software engineering, providing a visual representation of a system's structure by showcasing its various components and their interactions. These diagrams simplify complex systems, making it easier for developers to design, understand, and communicate the architecture.

1. Component

Represent modular parts of the system that encapsulate functionalities. Components can be software classes, collections of classes, or subsystems.

- **Symbol:** Rectangles with the component stereotype («component»).
- **Function:** Define and encapsulate functionality, ensuring modularity and reusability.

2. Interfaces

Specify a set of operations that a component offers or requires, serving as a contract between the component and its environment.

- **Symbol:** Circles (lollipops) for provided interfaces and half-circles (sockets) for required interfaces.
- **Function:** Define how components communicate with each other, ensuring that components can be developed and maintained independently.

3. Relationships

Depict the connections and dependencies between components and interfaces.

- **Symbol:** Lines and arrows.
    - **Dependency (dashed arrow):** Indicates that one component relies on another.
    - **Association (solid line):** Shows a more permanent relationship between components.
    - **Assembly connector:** Connects a required interface of one component to a provided interface of another.
- **Function:** Visualize how components interact and depend on each other, highlighting communication paths and potential points of failure.

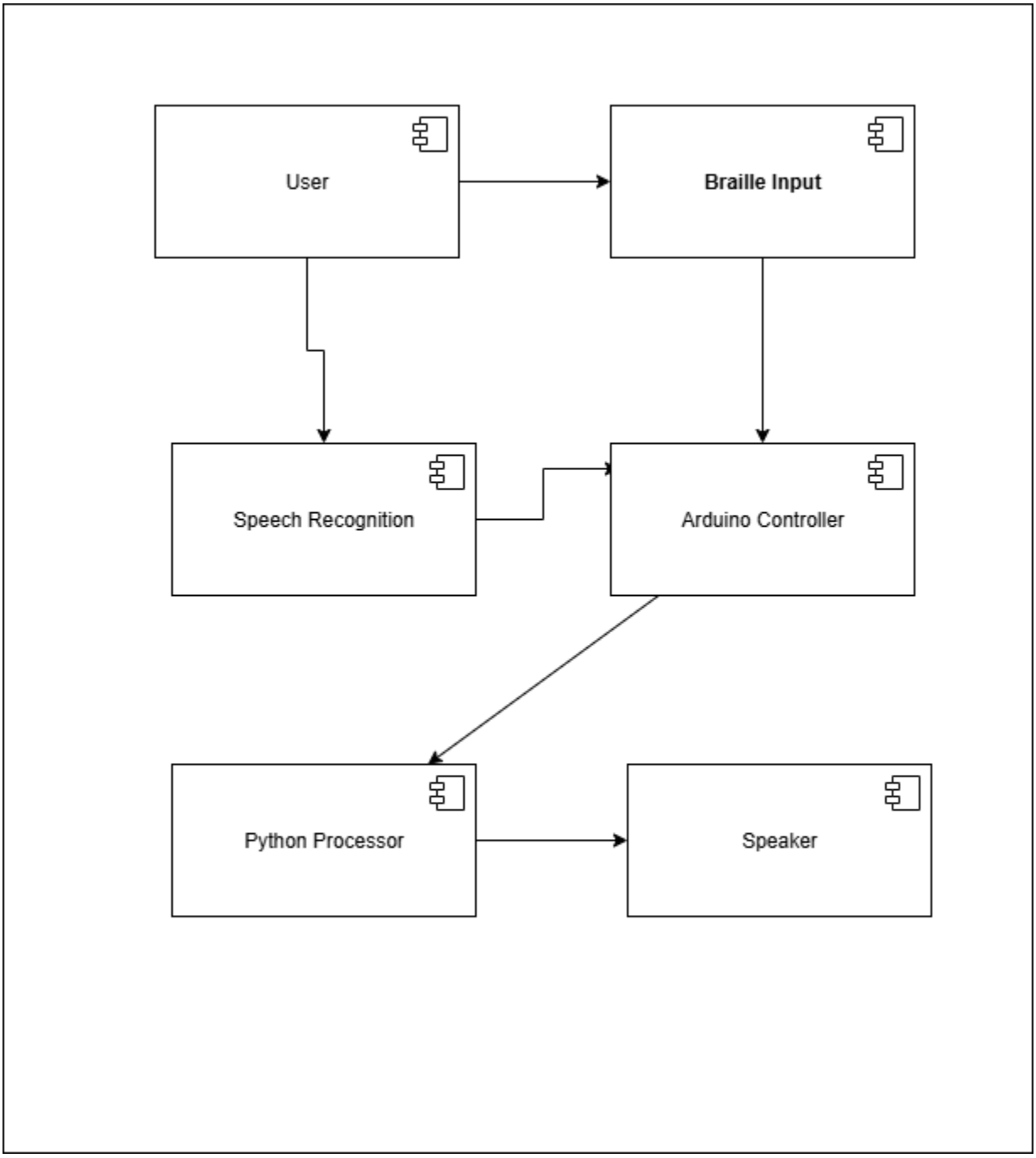



Figure 8: Component Diagram

### 3.3.9 Deployment Diagram

A Deployment Diagram is a type of Structural UML Diagram that shows the physical deployment of software components on hardware nodes. It illustrates the mapping of software components onto the physical resources of a system, such as servers, processors, storage devices, and network infrastructure.

- **Nodes**: These represent the physical hardware entities where software components are deployed, such as servers, workstations, routers, etc.
- **Components**: Represent software modules or artifacts that are deployed onto nodes, including executable files, libraries, databases, and configuration files.
- **Artifacts**: Physical files that are placed on nodes represent the actual implementation of software components. These can include executable files, scripts, databases, and more.
- **Dependencies**: These show the relationships or connections between nodes and components, highlighting communication paths, deployment constraints, and other dependencies.
- **Associations**: Show relationships between nodes and components, signifying that a component is deployed on a particular node, thus mapping software components to physical nodes.
- **Deployment Specification**: This outlines the setup and characteristics of nodes and components, including hardware specifications, software settings, and communication protocols.
- **Communication Paths**: Represent channels or connections facilitating communication between nodes and components and includes network connections, communication protocols, etc.

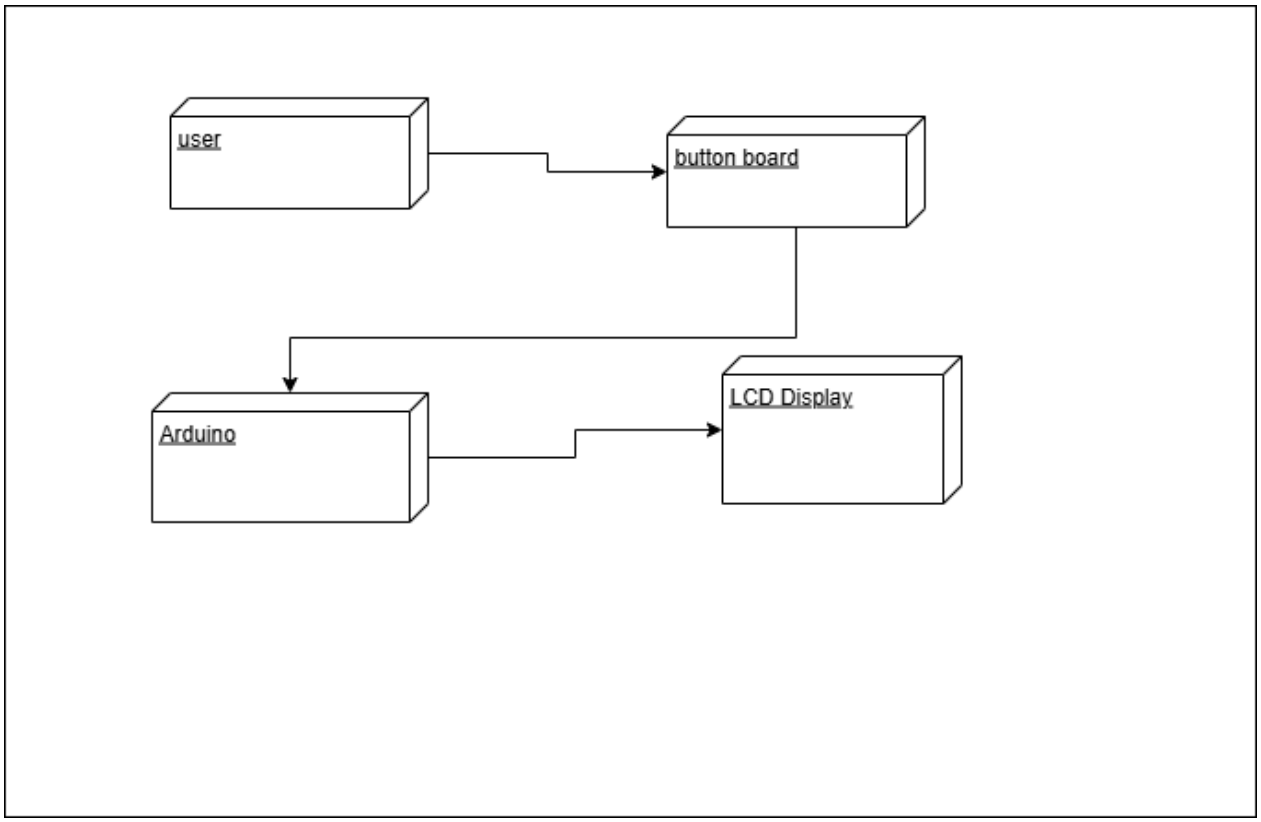
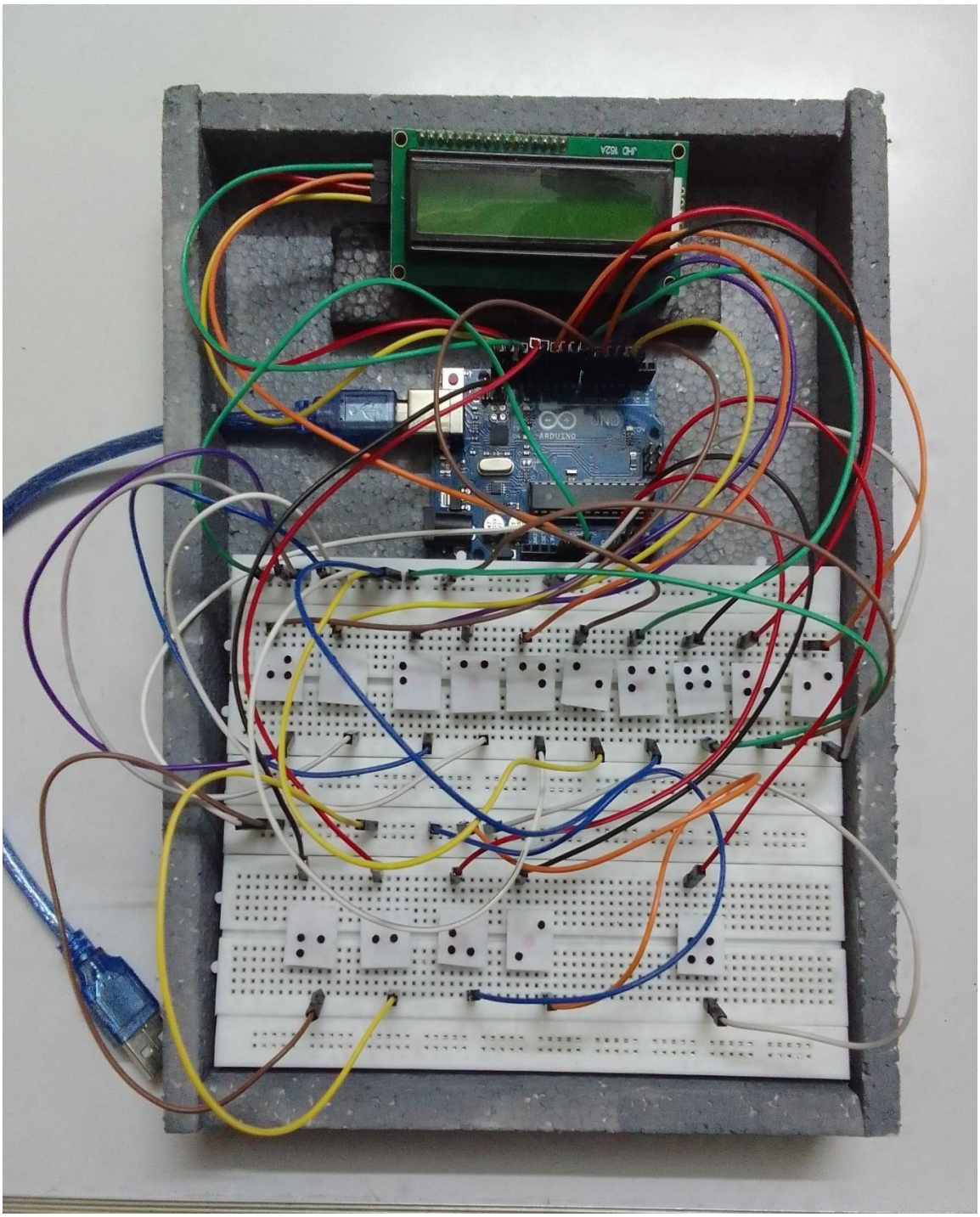



Figure 9: Deployment Diagram

## 3.4 User Interface




Press Enter to start speaking...
Listening... Speak now!
You said: 7 - 2
Math Expression: 7 - 2
Speaking: The result is 5
Press Enter to start speaking...
Listening... Speak now!
You said: 2 + 2
Math Expression: 2 + 2
Speaking: The result is 4
Press Enter to start speaking...

## 3.5 Test Case Design

1. Button Input Test Cases

| Test Case ID | Description | Input | Expected Output | Pass Criteria |
|---|---|---|---|---|
| TC-BTN-01 | Single digit entry | Press 5 | LCD: Expr: 5 | Digit displayed |
| TC-BTN-02 | Multi-digit entry | Press 1, 2 | LCD: Expr: 12 | Expression displayed |
| TC-BTN-03 | Operator entry | Press + | LCD: Expr: 12+ | Operator appended |
| TC-BTN-04 | Complete expression | Press 2 + 3 = | LCD: Result: 5, Voice: "The result is 5" | Result correct |
| TC-BTN-05 | Division by zero | Press 8 / 0 = | LCD: Error, Voice: "Error in calculation" | Error handled |

2. Voice Input Test Cases

| Test Case ID | Description | Voice Input | Expected Output | Pass Criteria |
|---|---|---|---|---|
| TC-VC-01 | Simple addition | "two plus three" | LCD: Result: 5, Voice: "The result is 5" | Result correct |
| TC-VC-02 | Square root | "square root of eighty one" | LCD: Result: 9, Voice: "The result is 9" | Special op handled |
| TC-VC-03 | Power | "two power three" | LCD: Result: 8, Voice: "The result is 8" | Power op handled |
| TC-VC-04 | Invalid speech | "calculate happiness" | LCD: Error, Voice: "Sorry, I did not understand" | Error handled |

## 3. Output Test Cases

| Test Case ID | Description | Input | Expected Output | Pass Criteria |
|---|---|---|---|---|
| TC-OUT-01 | LCD display works | Press `7` | LCD: `Expr: 7` | LCD updates |
| TC-OUT-02 | Voice output works | Press `6 / 2 =` | Voice: *"The result is 3"* | Voice speaks correctly |

## 4. Error Handling Test Cases

| Test Case ID | Description | Input | Expected Output | Pass Criteria |
|---|---|---|---|---|
| TC-ERR-01 | Invalid expression | Press `2 ++ 3 =` | LCD: `Error`, Voice: *"Error in calculation"* | Error shown |
| TC-ERR-02 | Hardware input bounce | Press `5` quickly | LCD: `Expr: 5` only once | No multiple entries |

## 5. Integration Test Cases

| Test Case ID | Description | Action | Expected Result |
|---|---|---|---|
| TC-INT-01 | Arduino → Python | Button input 3+4= | Python receives EXPR:3+4, evaluates 7, returns to Arduino |
| TC-INT-02 | Python → Arduino | Python result 9 | Arduino displays Result: 9 on LCD |
| TC-INT-03 | End-to-end flow | Voice input “five times five” | Python evaluates 25, Arduino LCD shows Result: 25, Voice output correct |

# CHAPTER 4: IMPLEMENTATION AND TESTING

## 4.1 Source Code:

### 4.1.1 Arduino Code:

```
void setup() {
  Serial.begin(9600);
  delay(2000);              // let Serial port settle (and allow Python to open)
  Serial.println("ARDUINO_READY");
}

void loop() {
  if (Serial.available()) {
    String line = Serial.readStringUntil('\n');
    line.trim();
    // echo back what we received
    Serial.print("RECV:");
    Serial.println(line);

    // a small handshake example
    if (line == "PING") {
    Serial.println("PONG");
    }
  }
  delay(10);
}
```

### 4.1.2 Python Code:

```
import time
import serial
import serial.tools.list_ports

def list_ports():
    ports = serial.tools.list_ports.comports()
    for p in ports:
        print(p.device, '-', p.description)
    return [p.device for p in ports]

ports = list_ports()
if not ports:
    print("No serial ports found. Plug in the Arduino and try again.")
    raise SystemExit

# choose port: either set manually or auto-pick a likely Arduino
# Replace 'COM3' with the port you saw in Arduino IDE, or pick the first from list
SERIAL_PORT = ports[0]  # or "COM3" or "/dev/ttyACM0"
BAUD = 9600

print("Opening", SERIAL_PORT)
ser = serial.Serial(SERIAL_PORT, BAUD, timeout=1)
time.sleep(2)  # wait for Arduino reset

# read any initial lines
print("Reading any startup lines...")
while ser.in_waiting:
```

```
    print("  < ", ser.readline().decode(errors='ignore').strip())

# send a PING and wait for PONG
ser.write(b"PING\n")
t0 = time.time()
while time.time() - t0 < 5:  # 5 second timeout
    if ser.in_waiting:
        line = ser.readline().decode(errors='ignore').strip()
        print("  < ", line)
        if line == "PONG":
            print("Handshake successful!")
            break
else:
    print("No PONG received. Check Arduino sketch and port.")

ser.close()
```

## Arduino code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
// Pins for 0–9, +, -, *, /, Enter (=)
const int buttonPins[15] = {
  2, 3, 4, 5, 6, 7, 8, 9, 10, 11,  // 0-9
  12, 13, A0, A1,                  // + - * /
  A2                           // Enter (=)
};
```

```
String symbols[15] = {
  "0","1","2","3","4","5","6","7","8","9",
  "+","-","*","/","="
};
String expression = "";

void setup() {
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Voice+Braille");

  for (int i = 0; i < 15; i++) {
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
}

void loop() {
  for (int i = 0; i < 15; i++) {
    if (digitalRead(buttonPins[i]) == LOW) {
      delay(600); // debounce

      if (symbols[i] == "=") {
        if (expression.length() > 0) {
          Serial.println("EXPR:" + expression);
```

```
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Solving...");
      lcd.setCursor(0, 1);
      lcd.print(expression);
      expression = "";
     }
    } else {
     expression += symbols[i];
     Serial.println("SAY:" + symbols[i]);

     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print("Expr:");
     lcd.setCursor(0, 1);
     lcd.print(expression);
    }
  }
}

// Receive result from Python
if (Serial.available()) {
  String incoming = Serial.readStringUntil('\n');
  if (incoming.startsWith("RESULT:")) {
  String res = incoming.substring(7);
  lcd.clear();
   lcd.setCursor(0, 0);
   lcd.print("Result:");
```

```
      lcd.setCursor(0, 1);
      lcd.print(res);
    }
  }
}
```

## Python Code:

```
import speech_recognition as sr
import pyttsx3
import serial
import time
import math

# Connect to Arduino (change COM3 to your port from Arduino IDE → Tools → Port)
arduino = serial.Serial("COM3", 9600, timeout=1)
time.sleep(2)  # wait for Arduino reset

# Setup recognizer and mic
recognizer = sr.Recognizer()
mic = sr.Microphone()

def speak(text):
    """Speak the given text using pyttsx3"""
    print(" Speaking:", text)
    engine = pyttsx3.init()  # reinitialize each time (fixes one-time bug)
    engine.setProperty("rate", 150)
    engine.setProperty("volume", 1.0)
```

```
    engine.say(text)
    engine.runAndWait()
    engine.stop()

def listen():
    """Capture voice input and return text"""
    with mic as source:
        print(" Listening... Speak now!")
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)

    try:
        text = recognizer.recognize_google(audio)
        print(" You said:", text)
        return text
    except sr.UnknownValueError:
        speak("Sorry, I did not understand")
        return None
    except sr.RequestError:
        speak("Speech recognition service unavailable")
        return None

def process_expression(spoken_expr):
    """Convert spoken words into a math expression"""
    expr = (spoken_expr.lower()
            .replace("plus", "+")
            .replace("minus", "-")
            .replace("times", "*")
```

```
        .replace("into", "*")
        .replace("multiplied by", "*")
        .replace("divide", "/")
        .replace("divided by", "/")
        .replace("mod", "%")
        .replace("power of", "**")
        .replace("to the power of", "**")
        .replace("power", "**")
        .replace("^", "**")   # fix: convert ^ to ** (Python exponentiation)
        .replace("square root of", "math.sqrt("))
    return expr

# ------------------ Main Loop ------------------
while True:
input("☞ Press Enter to start speaking...")
spoken_expr = listen()
    if spoken_expr:
        try:
          expr = process_expression(spoken_expr)
          print(" Math Expression:", expr)
          result = eval(expr, {"__builtins__": None, "math": math})
# Speak result
          speak(f"The result is {result}")
 # Send result to Arduino
          arduino.write(f"RESULT:{result}\n".encode())
        except Exception as e:
          print("Error:", e)
          speak("There was an error in calculation")
```

arduino.write(b"RESULT:Error\n")

## 4.1.3 Draw Functionality Module

Input Module

a) Voice Input Sub-Module

- Captures user's voice using a microphone.
- Converts speech into text using the Speech Recognition API (Google Speech or SpeechRecognition library**)**.
- Supports commands like:
  - "Two plus three"
  - "Square root of 81"
- Handles errors such as unclear speech or no input.

b) Braille Input Sub-Module

- Takes input through push buttons on Arduino, representing digits (0–9) and operators (+, −, ×, ÷, =).
- Sends the input sequence (expression) to Python through Serial Communication.
- Each button press is also spoken out loud using speech feedback for blind users.

Processing Module

a) Expression Conversion

- Converts spoken mathematical phrases into a valid Python expression (e.g., "2 to the power 3" → 2**3).
- Uses preprocessing to handle mathematical terms (plus, divide, square root, etc.).
- Supports decimal inputs and advanced math functions.

b) Computation Engine

- Evaluates the converted expression safely using Python's eval() with the math library.
- Supports:
  - Basic operations: +, -, *, /, %
  - Power: **
  - Square root: math.sqrt()
- Handles exceptions like invalid input or division by zero.

Output Module

a) LCD Display Output

- Arduino receives the computed result from Python through Serial communication.
- Displays result on a 16x2 I2C LCD in readable format (e.g., “Result: 25.6”).
- Clears screen automatically for next operation.

b) Voice Output

- Python uses pyttsx3 or Google Text-to-Speech API to speak the result aloud.
- Example: “The result is nine.”
- Provides dual feedback (auditory + visual).

Communication Module

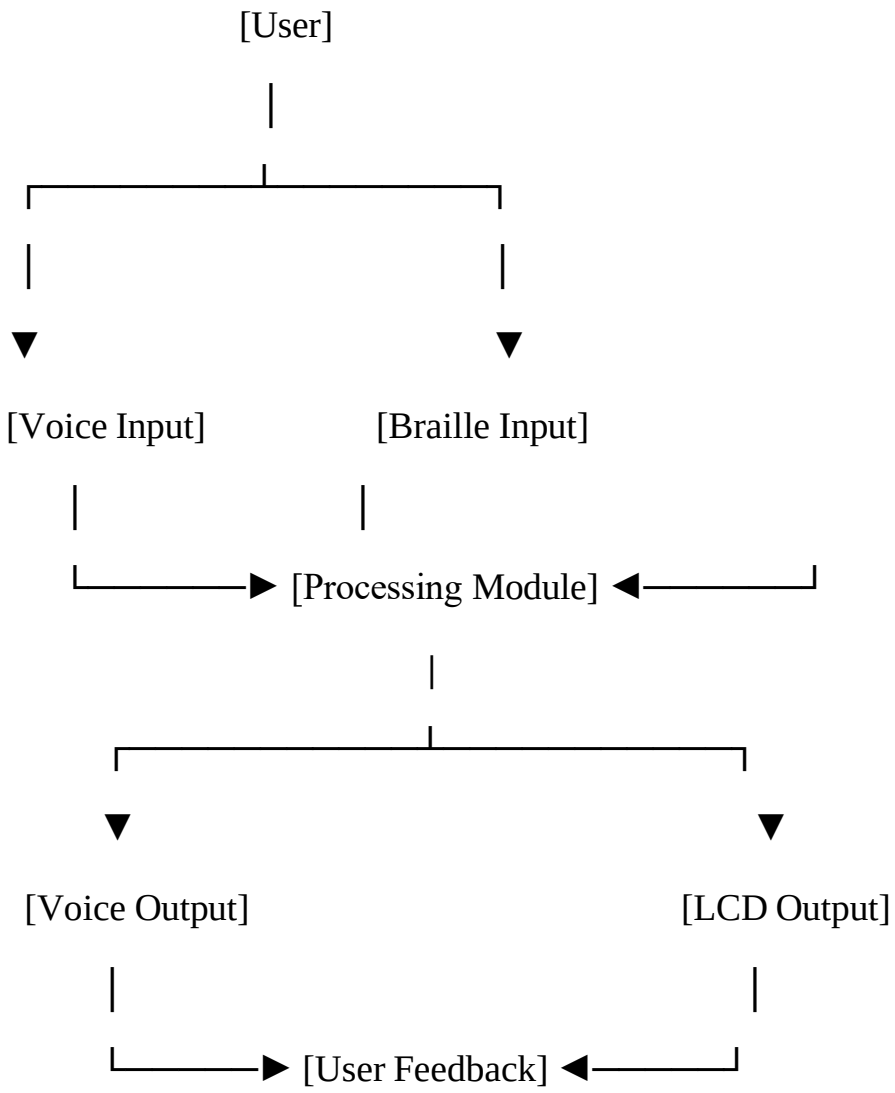
- Serial Communication (Arduino ↔ Python)
    - Transfers expressions and results between Arduino and Python.
    - Baud rate: 9600 bps.
    - Data format examples:
        - From Arduino → Python: EXPR:2+3
        - From Python → Arduino: RESULT:5

Power & Hardware Interface Module

- Supplies stable 5V power to Arduino and components.
- LCD connected via I2C (A4 = SDA, A5 = SCL).
- Braille buttons connected to digital pins with pull-down resistors.
- Optional solar/battery module for portable power supply.

Error Handling & Feedback Module

- Detects and announces invalid expressions, syntax errors, or microphone issues.
- Ensures:
    - Voice: “Error in calculation. Please try again.”
    - LCD: Displays “Error”.
- Prevents system from crashing during invalid input.


[User]
[Voice Input]
[Braille Input]
[Processing Module]
[Voice Output]
[LCD Output]
[User Feedback]

## 4.2 Testing Approach

### 4.2.1 Unit Testing

**1. Arduino Button Input Module:** Verify each button correctly registers and sends the expected signal.

**2. Arduino LCD Display Module**: Check whether LCD displays text correctly.

**3. Serial Communication (Arduino ↔ Python):** Ensure data is correctly transmitted between Arduino and Python.

**4. Python Expression Evaluation Module**: Test calculation logic.

**5. Speech Recognition Module (Python)**: Ensure voice input is correctly recognized.

**6. Text-to-Speech Module (Python)**: Verify result is spoken clearly.

### 4.2.2 Integrate Testing

1. **Button Input → Arduino → Python → LCD**

- **Steps**:
    1. Press 1, +, 2, then =.
    2. Arduino sends "1+2" to Python.
    3. Python evaluates and sends "RESULT: 3".
    4. Arduino displays "Result: 3" on LCD.
- **Expected Result**: LCD shows Result: 3.

2. **Button Input → Arduino → Python → Voice Output**

- **Steps**:
    1. Press 4, *, 5, then =.
    2. Arduino sends "4*5".
    3. Python evaluates as 20.
    4. Python TTS says "The result is twenty".
- **Expected Result**: Voice output matches calculation.

**3. Voice Input → Python → Arduino → LCD**

- **Steps**:
    1. User says: “Square root of 81”.
    2. Python recognizes "sqrt(81)".
    3. Python evaluates as 9.
    4. Sends "RESULT: 9" to Arduino.
    5. Arduino displays on LCD.
- **Expected Result**: LCD shows Result: 9.

**4. Voice Input → Python → Voice Output**

- **Steps**:
    1. User says: “Two power three”.
    2. Python recognizes "2^3".
    3. Python evaluates as 8.

4. Python TTS says: "The result is eight".

- **Expected Result**: Speaker announces correct result.

**5. Error Handling (Invalid Input)**

- **Steps**:
    1. Press random buttons like +, =, or say “Divide by zero”.
    2. Arduino/Python send expression "8/0".
    3. Python detects division by zero.
    4. Returns "Error" → LCD shows Error.
    5. TTS says "Invalid input, please try again".
- **Expected Result**: Both LCD and voice show error message.

**6. Mixed Input (Button + Voice)**

- **Steps**:
    1. Enter part of expression with buttons (e.g., 5+).
    2. Speak rest of it (e.g., “seven”).
    3. System builds full expression "5+7".
    4. Python evaluates result.
- **Expected Result**: Works seamlessly across input modes.

## 4.2.3 Beta Testing

1. **Identify Test Users**

- Select visually impaired students and teachers/assistants from blind schools.

- Include a mix of:
    - Beginner users (new to calculators).
    - Experienced users (already use talking calculators).

2. **Prepare Test Environment**

- Hardware setup: Arduino + LCD + Buttons + Microphone + Speaker.
- Place project on a portable board (wood/acrylic).
- Power: USB / Battery / Solar (if implemented).

3. **Test Scenarios for Beta Users**

1. **Basic Operations**
    - Press 2 + 3 = → Expect "5" on LCD + speech.
    - User confirms correctness.
2. **Voice Operations**
    - Say: *"*Six divided by three*"* → Expect "2".
    - User checks both LCD and audio.
3. **Advanced Operations**
    - Say: *"*Square root of 81*"* → Expect "9".
    - Say: "Two power five" → Expect "32".
4. **Error Handling**
    - Say: "Divide by zero".
    - Expect error message on LCD and voice prompt.
5. **Accessibility Check**

- Buttons should be easy to press.
- Braille markings should be clear.
- Audio should be loud and understandable.

## 4.3 Test Case

| Test Case ID | Module | Test Description | Input | Expected Output | Actual Output | Status |
|---|---|---|---|---|---|---|
| TC-01 | Voice Input | Speech-to-text conversion | Speak: "5 plus 3" | Recognized as 5+3 | 5+3 detected | Pass |
| TC-02 | Voice Input | Unsupported phrase | Speak: "Hello world" | Error: "Invalid input" | Error shown | Pass |
| TC-03 | Braille Input | Digit button press | Button 1 → 1 | LCD shows 1, voice says "One" | Works correctly | Pass |
| TC-04 | Braille Input | Operator button press | Button + → + | LCD shows +, voice says "Plus" | Works correctly | Pass |
| TC-05 | Expression Evaluation | Simple addition | 2+2 | 4 | 4 | Pass |
| TC-06 | Expression Evaluation | Division | 10/2 | 5 | 5 | Pass |
| TC-07 | Expression Evaluation | Division by zero | 5/0 | Error: "Division by zero" | Error spoken & displayed | Pass |
| TC-08 | Expression Evaluation | Power function | 2^3 | 8 | 8 | Pass |
| TC-09 | Expression Evaluation | Square root | sqrt(81) | 9 | 9 | Pass |
| TC-10 | Output (LCD) | Check display output | 7*6 | LCD shows 42 | LCD displays correctly | Pass |
| TC-11 | Output (Voice) | Spoken result | Result 42 | Voice says: "The result is forty two" | Correct speech | Pass |

| Test Case ID | Module | Test Description | Input | Expected Output | Actual Output | Status |
|---|---|---|---|---|---|---|
| TC-12 | Arduino-Python Communication | Serial link test | Send "RESULT:10" | LCD shows "Result: 10" | Displays correctly | Pass |
| TC-13 | Error Handling | Invalid math input | 5++2 | Error: "Invalid input" | Error handled correctly | Pass |
| TC-14 | Multi-Calculation | Sequential inputs | Perform 2+3 then 4*2 | Results update after each | LCD & voice update correctly | Pass |
| TC-15 | Voice Mode Trigger | Trigger voice input | Press "Voice Mode" button | Python listens & calculates | Works fine | Pass |

# CHAPTER 5: CONCLUSION

## 5.1 Conclusion

The Voice + Braille Calculator has been successfully designed and implemented as an assistive technology tool for visually impaired users. The system provides dual input modes (buttons with Braille markings and voice commands) and dual output modes (audio through text-to-speech and visual feedback on LCD), making it accessible to both blind and sighted users. Through the integration of Arduino hardware (buttons, LCD) and Python software (speech recognition, expression evaluation, text-to-speech), the project demonstrates how embedded systems and AI can be combined to create inclusive and user-friendly applications. The calculator not only performs basic arithmetic operations (addition, subtraction, multiplication, division) but also supports advanced functions like percentage, square root, and power calculations. Error handling ensures that invalid inputs (e.g., division by zero) are managed gracefully with proper user feedback. In conclusion, this project demonstrates the potential of low-cost hardware and open-source software in addressing accessibility challenges, ultimately empowering visually impaired students to perform mathematical operations independently and confidently.

## 5.2 Limitation of The System

1. **Dependency on Laptop/PC**
    - Voice recognition and text-to-speech currently run on Python (PC/laptop).
    - This makes the system less portable compared to standalone calculators.
2. **Limited Speech Recognition Accuracy**
    - Background noise or unclear pronunciation may lead to incorrect recognition.
    - Only basic math vocabulary (numbers, operators, sqrt, power, percent) supported.
3. **Restricted Button Count**
    - Only digits (0–9) and basic operators (+, −, ×, ÷, =) are available.
    - Complex math symbols (log, sin, cos, etc.) are not supported in current design.
4. **Error Handling Constraints**
    - While division by zero and invalid input are handled, other advanced error cases are limited.
5. **Hardware Size & Layout**

- Breadboard-based setup is bulky and not suitable for daily portable use.
- Buttons may be small and difficult for some visually impaired users to press.

6. **No Built-in Power Backup**
    - System depends on continuous power from laptop/PC via USB.
    - No battery or solar backup included in the basic version.

## 5.3 Future Scope of The System

**Standalone Device (Without Laptop/PC)**

- Replace the dependency on Python running on a laptop with an embedded voice recognition module (like Elechouse VR3/VR4 or ESP32 with offline AI libraries).
- This would make the calculator portable and self-contained.

**Solar and Battery Powered**

- Add rechargeable battery or solar charging support to make it usable in rural schools and areas with poor electricity.
- Increases portability and independence.

**Improved Speech Recognition**

- Add support for multiple languages (English, Hindi, Marathi, etc.) for wider accessibility.
- Use AI-based speech recognition models for better accuracy in noisy environments.

**Expanded Mathematical Functions**

- Extend beyond basic operations to scientific calculator features like:
    - Trigonometric functions (sin, cos, tan).
    - Logarithms and exponentials.
    - Fractions and matrix operations.
- This would benefit high school and college students.

**Braille Output via Servo Motors**

- Implement servo-motor-driven tactile Braille output pins that raise/lower dots to display results physically.
- This would allow blind users to feel results in addition to hearing them.

**Compact and User-Friendly Design**

- Move from breadboard to custom PCB design with proper casing.
- Larger tactile buttons with clear Braille embossing for easier input.

**Result History and Memory**

- Store last few calculations in EEPROM or SD card.
- Voice feedback like: *"*Last result was 25*"*.

**Integration with Learning Tools**

- Add a Math Tutor Mode where the calculator not only gives the result but also explains steps**.**
- Example: For 2+3, it could say "Adding 2 and 3 gives 5".

**Wireless Connectivity**

- Use Bluetooth/Wi-Fi module to connect with smartphones or tablets.
- Could sync results to a mobile app for teachers/assistants.

**Wider Testing and Deployment**

- Conduct trials in schools for the blind.
- Collect large-scale feedback to refine usability, button design, and language support.

# REFERENCES


### Books & Research Papers

- Raj Kamal, Embedded Systems: Architecture, Programming and Design, McGraw-Hill.
- Banzi, Massimo, Getting Started with Arduino, Maker Media, 2014.
- Z. Bujacz, R. Strumillo, "Accessible Interfaces for Visually Impaired People", IEEE Multimedia, 2012.
- K. Miesenberger, J. Klaus, W. Zagler, Computers Helping People with Special Needs, Springer, 2016.

### Arduino Documentation

- Arduino Official Website: https://www.arduino.cc
- LiquidCrystal_I2C Library Documentation: https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

### Python Documentation & Libraries

- SpeechRecognition Library: https://pypi.org/project/SpeechRecognition/
- pyttsx3 (Text-to-Speech): https://pypi.org/project/pyttsx3/
- PySerial: https://pythonhosted.org/pyserial/

### Assistive Technology References

- World Health Organization (WHO), "World Report on Vision", 2019.
- American Foundation for the Blind (AFB): https://www.afb.org

### Online Resources & Tutorials

- Tinkercad Circuits: https://www.tinkercad.com/circuits
- Instructables, "Talking Calculator with Arduino and Python".
- Stack Overflow & Arduino Forum (for debugging and troubleshooting).

# GLOSSARY

| Term | Definition |
|---|---|
| **Arduino Uno** | A microcontroller board used to read Braille button inputs and display results on an LCD. It acts as the hardware control unit of the project. |
| **Python** | The main programming language used for processing voice input, performing calculations, and communicating with Arduino. |
| **Speech Recognition** | Technology that converts spoken words into text using a microphone and APIs like Google Speech Recognition. |
| **Text-to-Speech (TTS)** | Converts textual results into spoken voice output. Implemented using libraries like `pyttsx3` or Google Cloud TTS. |
| **Braille Input** | A set of tactile buttons representing numbers and mathematical operators, allowing visually impaired users to input expressions. |
| **LCD Display** | A 16x2 character display used to show results and messages for both blind and sighted users. |
| **Serial Communication** | The data exchange process between Arduino and Python using USB (COM port) at 9600 baud rate. |
| **Microphone** | An input device that captures voice commands for the calculator. |
| **Speaker** | An output device that plays the spoken results generated by the Text-to-Speech engine. |
| **Math Library (`math`)** | A Python library providing advanced mathematical functions like square root, power, trigonometric, and logarithmic calculations. |
| **Expression Evaluation** | The process of converting user input (voice or Braille) into a mathematical expression and calculating the result. |
| **Error Handling** | Detects invalid expressions or speech and provides user-friendly feedback through LCD and voice. |
| **Voice Mode** | A mode in which the calculator accepts spoken input instead of Braille button input. |
| **Power Supply / Battery** | Provides 5V regulated power to Arduino, LCD, and other components. Optionally solar-powered for portability. |

| Term | Definition |
| --- | --- |
| **Google Cloud API** | External service used for accurate speech recognition and natural-sounding voice output (optional upgrade). |
| **Pyttsx3** | A Python library for offline Text-to-Speech that converts calculation results into audible speech. |
| **Gantt Chart** | A graphical tool used to represent the timeline and phases of the project. |
| **UML Diagram** | Unified Modeling Language diagrams like use case, activity, class, sequence, and component diagrams used for software design representation. |
| **Testing** | The phase in which different modules (voice, Braille, LCD, serial communication) are verified for correct performance. |