A PROJECT REPORT

ON

“Smart Blind Stick for Visually Impaired”

Submitted in partial fulfillment of

B.Tech Computer Science and Engineering.

Session: 2025-2026

DEPARTMENT OF *COMPUTER SCIENCE AND ENGINEERING*

G.C.R.G. Group of Institutions

Submitted To:

Er. ……

Submitted By:

Utkarsh Vishwakarma (22047301000

Vipin Yadav (2204730100064)

Shivam Ranjan (22047301000

Chandan Gupta (22047301000

Anubhav Pandey (22047301000

Divyanshu Shukla (22047301000

CERTIFICATE

This is certified that Project entitled "**Smart Blind Stick for Visually Impaired**" which is submitted by VIPIN YADAV, UTKARSH VISHWAKARMA, SHIVAM RANJAN, CHANDAN GUPTA, ANUBHAV PANDEY, DIVYANSHU SHUKLA of B.Tech 4th year, Computer Science & Engineering, G.C.R.G. Group of Institutions, Lucknow, for the award of the B.Tech, is a bonafide record of work carried out by them under guidance of **Er ……**

The content of this project has not been submitted to any university or institute for award of any degree or diploma.

_ _ _ _ _ _ _ _ _ _ _ _ _

Er.
Head of department
(Project guide)

_ _ _ _ _ _ _ _ _ _ _ _ _

External Examiner

Date: _ _/ _ _/ _ _ _ _

Place: Lucknow

SELF ATTESTATION

This is certifying that, we have personally worked on the dissertation entitled " **Smart Blind Stick for Visually Impaired** **".** A case study and data mentioned in this report was obtained during genuine work done and collected by us.

Any other data and information in this report, which has been collected from outside agency, has been duly acknowledged.

STUDENTS:

Vipin Yadav

Shivam Ranjan

Chandan Gupta

Anubhav Pandey

Divyanshu Shukla

Utkarsh Vishwakarma

ABSTRACT

In this era of technological advancements, there is a growing need to develop innovative solutions to enhance the independence and safety of visually impaired individuals. In response to this, we present a Smart Blind Stick, a comprehensive assistive device integrating a variety of sensors and communication technologies to aid navigation and environmental perception for the visually impaired.

God gifted sense of vision to the human being is an important aspect of our life. But there are some unfortunate people who lack the ability of visualizing things. The visually impaired have to face many challenges in their daily life. The problem gets worse when there is an obstacle in front of them. Blind stick is an innovative stick designed for visually disabled people for improved navigation. We are proposing a low-cost walking stick based on internet of things for efficient interface to blind people.

In this paper, we are build this smart blind stick for blind peoples, because of sometimes accident was created and also they are facing many problems like that so we are using ultra sonic sensor here also this project is base on Arduino Uno

OVERVIEW

Describe the problem of safe navigation for visually impaired people and state that the project develops a smart blind stick using ultrasonic sensors, ESP32, vibration feedback, and voice alerts to detect obstacles and guide the user.

The Smart Blind Stick incorporates an array of sensors including ultrasonic and moisture sensors to detect obstacles and changes in the environment, ensuring safe navigation in various conditions. Bluetooth connectivity enables seamless communication with a smartphone or other devices, facilitating real-time data exchange and user interaction. The integration of a speaker and buzzer provides auditory feedback and alerts, while a vibrator sensor offers tactile feedback to the user, enhancing situational awareness.

PROFILE OF PROBLEM ASSIGNED

a. Introduction to project:

Explain what a smart blind stick is and why assistive navigation devices are needed. Mention key features like obstacle detection, vibration, and speech output.

The Smart Blind Stick is an innovative assistive device engineered to empower visually impaired individuals with enhanced mobility and safety. By integrating state-of-the-art sensors including ultrasonic and moisture sensors, alongside communication technologies such as Bluetooth, GSM, and GPS, this device offers real-time environmental perception and connectivity. Through the utilization of auditory and tactile feedback mechanisms, the Smart Blind Stick provides users with vital information about obstacles and changes in their surroundings, fostering greater independence and confidence in navigation. This project signifies a significant advancement in assistive technology, addressing the critical need for effective solutions to support the daily lives of visually impaired individuals.

The Smart Blind Stick project aims to bridge existing gaps in traditional mobility aids by offering a comprehensive solution that leverages modern technologies. With its compact design and user-friendly interface, the device promises to revolutionize the way visually impaired individuals interact with their environment. By combining advanced sensor systems with connectivity features, it enables seamless communication and real-time assistance, ultimately empowering users to navigate their surroundings with greater ease and autonomy. As a result, the Smart Blind Stick stands as a testament to the potential of technology to improve accessibility and inclusivity for individuals with visual impairments.

b. b.Existing system:

Briefly discuss traditional white canes and simple electronic sticks that only give basic beeps or vibrations and their limitations (no direction info, limited range).

Currently, visually impaired individuals heavily rely on traditional white canes or guide dogs for navigation assistance. Additionally, existing systems may lack integration with communication technologies for remote assistance or navigation aids for complex environments.

Current State-of-Art Solutions

1. **Traditional White Cane:**
   - Provides tactile feedback through direct contact
   - Range: 0â€“2 meters (arm length + cane length)
   - User-dependent skill and training required
   - No directional specificity beyond left/right sweeping
   - No electronic feedback

2. **Simple Electronic Canes (Commercial):**
   - Typically use single ultrasonic sensor or IR emitter
   - Emit only beep or simple vibration when obstacle detected
   - Distance graduated into 2â€“3 levels only
   - No directional information
   - Limited battery life (5â€“8 hours)
   - High cost (â‚¹5,000â€“â‚¹15,000 INR)

3. **Research Prototypes:**
   - Some use computer vision or LIDAR for better accuracy
   - Require significant processing power and high cost
   - Not yet commercially viable or affordable for most users

Limitations of Existing Systems:
- Single or limited sensing points â†’ poor spatial awareness
- No voice feedback â†’ depends on audio learned by user
- Fixed sensitivity â†’ poor adaptability to different environments
- Limited battery optimization
- Bulky design, difficult to integrate with standard canes

c. Proposed system:

Smart Blind Stick Architecture:

The proposed system introduces a **modular**, **intelligent** **approach** that combines proven sensor technologies with modern embedded systems:

Sensing Layer:--

- Three HC-SR04 ultrasonic sensors (front, left, right) for 360Â° awareness in horizontal plane
- Range: 2 cm to 400 cm with high accuracy
- Polling frequency: ~8 Hz per sensor

Processing Layer:--

- ESP32 microcontroller as primary controller
- Reads all sensor inputs at high speed
- Implements decision logic for threat assessment
- Controls vibration motor with PWM intensity modulation
- Communicates with voice unit via UART serial protocol
- Manages mode switching (Indoor/Outdoor)
- Low power consumption (~80 mA active)

Feedback Layer:--

- Vibration Motor (**Coin** **ERM**): Provides haptic feedback with intensity levels (weak, medium, strong) based on distance
- Text-to-Speech (TTS) on Raspberry Pi Zero W: Generates natural voice alerts ("Obstacle ahead 50 centimeters") using espeak library
- Optional Buzzer: Low-battery warning and mode confirmation beeps

Power Layer:--

- Single 18650 Li-ion cell (3.7V nominal, 2600 mAh typical) or two in parallel
- TP4056 charging board with USB-C input
- LM2596 buck converter for 5V regulation
- Estimated runtime: 6â€“8 hours of continuous operation

Communication:--

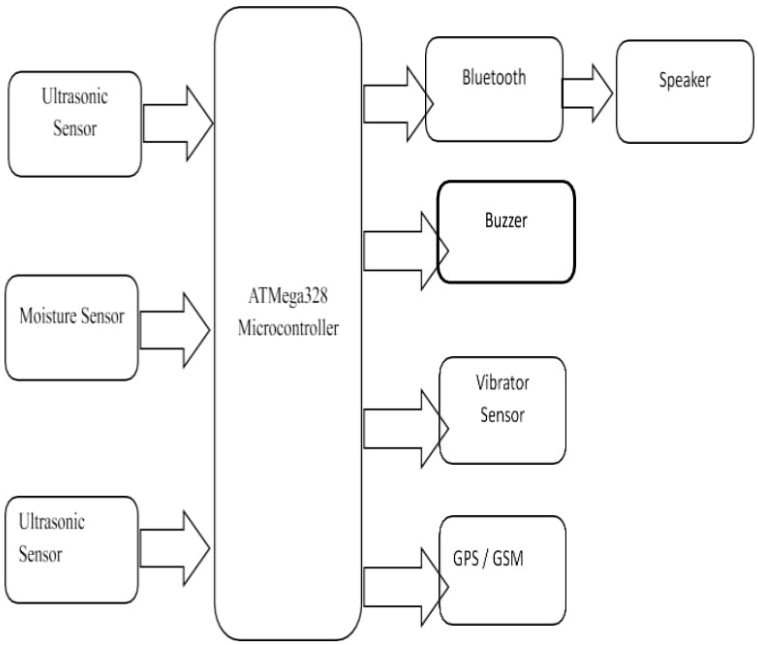
- Serial UART (115200 baud) between ESP32 and Raspberry Pi
- Compact JSON-like message format to minimize bandwidth

User Interface:--

- Power button (On/Off/Low Battery indication)
- Mode button (Indoor/Outdoor sensitivity toggle)

- Test button (system self-test)
- Simple audible feedback for mode changes

BLOCK DIAGRAM



ModulesL:-

Objective:

List objectives such as improving obstacle awareness, reducing collisions, keeping the device low-cost and portable, and supporting indoor/outdoor modes.

Primary Objectives:

1. Enhance Obstacle Awareness: Detect and communicate presence, direction, and distance of obstacles within a practical range (up to 4 meters).

2. Reduce Collision Risk: Provide timely alerts to enable users to adjust course before impact, significantly reducing trip and collision incidents.

3. Improve User Confidence: Enable visually impaired individuals to navigate independently in familiar and unfamiliar environments with greater confidence.

4. Affordability: Design a system with a bill of materials cost under â‚¹3,000â€“â‚¹4,000 INR, making it accessible to a broader user population compared to commercial alternatives.

5. Portability: Integrate seamlessly with standard walking canes; lightweight (<500 g), weather-resistant, and suitable for daily carry.

6. Adaptability: Implement switchable sensitivity modes to accommodate both indoor navigation (shops, corridors) and outdoor travel (open spaces, roads).

7. Reliability: Achieve robust operation with minimal false positives/negatives, tested across varied conditions (different surfaces, lighting, weather).

8. Accessibility: Create comprehensive documentation and open-source code to facilitate adoption and modification by makers, rehabilitation centers, and researchers

3. METHOLODY

The Smart Blind Stick system is decomposed into six functional modules:

Module 1: **Sensing Module**

Purpose Detect obstacles at various distances and directions
Components:
- 3Ã— HC-SR04 Ultrasonic Sensors (front, left, right)
- Trigger and Echo processing pins

Operation:
- Sends a 40 kHz ultrasonic pulse
- Measures echo return time
- Calculates distance = (time Ã— speed of sound) / 2
- Operating range: 2 cm â€“ 4 m
- Update frequency: ~8 Hz per sensor

Output: Distance values (in cm) to Processing Module

Module 2: **Processing** & Decision Logic Module

PURPOSE Interpret sensor data and make real-time decisions
Components:
- ESP32 DevKit microcontroller
- GPIO pins for sensor input
- UART interface for communication

Operation:
- Continuously reads three sensors in a loop
- Filters noisy sensor readings with simple averaging
- Implements threat assessment logic based on distance thresholds:
  - Very Close (<30 cm): Imminent collision risk â†’ maximum vibration + urgent voice alert
  - Near (30â€“80 cm): Approach hazard â†’ graduated vibration + measured voice alert
  - Far (80â€“200 cm): Awareness distance â†’ subtle vibration + directional notification
- Identifies which sensor has closest reading (front/left/right) to determine obstacle direction
- Enforces minimum gap between TTS calls (~1.2 seconds) to avoid alert saturation
- Manages mode-based threshold adjustments

Output: Vibration PWM signal + Serial message to Voice Module

---

Module 3: Vibration Feedback Module

Purpose: Provide haptic feedback to user
Components:
- Coin vibration motor (ERM) 3V 100 mA
- N-channel MOSFET (2N7002 or similar) driver
- Flyback diode for back-EMF protection
- PWM output from ESP32

Operation:
- Receives PWM signal (0â€“255) from Processing Module
- Translates PWM value to motor speed:
  - 0â€“50: Off or very light vibration
  - 50â€–150: Medium vibration (periodic pulses)
  - 150â€–255: Strong continuous vibration
- Vibration patterns indicate urgency and distance:
  - Continuous strong: Imminent hazard (very close)
  - Pulsed medium: Approaching hazard (near)
  - Brief light: Distant awareness (far)

Advantage:--User learns vibration intensity/pattern without needing to listen; works in noisy environments.

Module 4: Voice Alert Module (Text-to-Speech)

Purpose: Generate natural speech for directional and distance information
Components:
- Raspberry Pi Zero W (or ESP32-S3 as alternative)
- USB speaker or 3.5mm audio jack with small speaker
- Python runtime with pyserial and espeak libraries
- UART connection to ESP32

Operation:
- Listens on serial port for concise messages from ESP32 (format: `TTS,LEVEL,DIRECTION,DISTANCE`)
- Parses message and generates human-readable phrase:
  - Example: `TTS,VERY_CLOSE,left,25 at "Very close obstacle on the left, 25 centimeters"
- Calls espeak TTS engine to synthesize and play audio
- Enforces minimum repeat interval (~1.2 seconds) to prevent overlapping speech
- Alternative: Pre-recorded audio samples for higher quality

Output: Audio alert via speaker

Module 5: Power Management Module

Purpose: Provide stable**, regulated power to all components**
Components:
- 18650 Li-ion battery (3.7V, 2600 mAh) single or parallel configuration
- TP4056 lithium charging board with protection
- LM2596 buck converter (3.7V to 5V) or boost converter as needed
- Schottky diodes for reverse polarity protection
- Low battery detection circuit (voltage divider on ADC pin)

Operation:
- Accepts 5V input from external USB charger via TP4056
- Charges battery safely with over-current and over-voltage protection
- Provides regulated 5V output to ESP32, Pi, sensors, and motor
- Monitors battery voltage continuously via ADC
- Triggers low-battery alert when voltage drops below threshold (typically 3.0V nominal = ~2.8V per cell)
- Estimated runtime: 6 to 8 hours continuous operation

Module 6: User Interface Module

Purpose: Allow user interaction and system status indication
Components:
- Power button (momentary push)
- Mode button (momentary push)
- Test button (momentary push)
- Status buzzer (optional, for feedback)
- LED indicator (optional, for battery status)

Operation:
- Power button: Turns system on/off; held for 2 seconds to force shutdown
- Mode button: Toggles between Indoor and Outdoor sensitivity:
  - Indoor: More sensitive, shorter thresholds, frequent TTS
  - Outdoor: Less sensitive, longer thresholds, reduced false alerts
- Test button: Triggers self-test routine; reports sensor functionality and battery status via voice or buzzer
- Buzzer: Single beep = mode toggle confirmed; repeated beeps = low battery warning

---

Technology Used

Hardware Platform:

| Component | Choice | Rationale |

| Microcontroller | ESP32 DevKit | Dual-core, 240 MHz, 520 KB SRAM, built-in Wi-Fi/BLE, GPIO-rich, excellent for real-time sensor reading and PWM control. Arduino Nano is alternative for simpler designs but lacks memory for advanced features. |

| **Voice** Unit | Raspberry Pi Zero W | Dedicated TTS processing, offline espeak library for reliability, GPIO for peripherals, Linux environment for flexibility. ESP32-S3 with I2S DAC is more integrated alternative. |

| Sensors | HC-SR04 Ultrasonic | Affordable (~â‚¹100 each), well-documented, reliable outdoors/indoors, 2â€“400 cm range, low power consumption. LIDAR (VL53L1X) is superior but 5Ã— costlier. |

| Vibration Motor | Coin ERM (3V, 100 mA) | Compact form factor for cane handle integration, responsive haptic feedback, low current draw, comfortable intensity profile. Linear motors are smoother but larger. |

| Battery | 18650 Li-ion + TP4056 | High energy density, rechargeable, protection built into TP4056 board, standard in maker community. LiPo pouches are lighter but require custom protection circuits. |

| Voltage Regulation | LM2596 (Buck) | Efficient DC-DC buck converter, adjustable output, widely available, proven reliability in embedded systems. |

Software Stack:

Layer	Technology	Reason
MCU Firmware	Arduino IDE + C/C++	Standard for ESP32 development, extensive ultrasonic and PWM libraries, fast compile-upload cycle, large community support.

| Voice Engine | Raspberry Pi OS (Debian) + espeak (CLI) + Python 3 | Offline TTS (no cloud dependency), natural pronunciation, open-source, zero licensing cost, Python easy to modify and debug. |

| Serial Communication | UART at 115200 baud | Standard MCU-to-Pi link, simple protocol, low latency, no additional hardware overhead beyond wiring. |

| Testing Tools | Arduino Serial Monitor, Python console, oscope/multimeter | Essential for debugging sensor readings, verifying PWM signals, and monitoring power draw. |

Integration Approach:

1. Modular Design: Each module operates independently with well-defined interfaces (serial messages, GPIO signals), enabling parallel development and easier troubleshooting.

2. Event-Driven Architecture: ESP32 continuously polls sensors and updates outputs; Raspberry Pi reacts to incoming serial messages, avoiding blocking waits.

3. Fail-Safe Operation: If Raspberry Pi connection drops, vibration motor still functions independently; if vibration motor fails, voice alerts continue.

4. Real-Time Constraints: Sensor reading cycle time ~120 ms ensures latency <200 ms from obstacle entry to user alert, compatible with safe walking speed (~1 m/s).

4. Requirement Analysis

4.1 Hardware Specification

Primary Microcontroller: ESP32 DevKit
- Processor:Dual-core Xtensa 32-bit, 240 MHz
- Memory: 520 KB SRAM, 4 MB Flash
- I/O Pins:30+ GPIO, SPI, I2C, UART, ADC (12-bit)
- Power: 80â€“160 mA active, 10 ÂµA deep sleep
- Operating Voltage: 3.3V (internal), accepts 5V USB input
- Dimensions: 48 mm Ã— 25 mm
- **Cost:** ~â‚¹600 INR

Why: Superior to Arduino Nano due to higher clock speed (2Ã— faster), more RAM for buffering sensor data, native PWM support, and easier serial-to-Raspberry Pi integration.

Voice Unit: Raspberry Pi Zero W

- Processor: Single-core ARM, 1 GHz
- Memory: 512 MB RAM, microSD card storage (minimum 8 GB)
- I/O: GPIO, UART, SPI, I2C, USB
- Power: 100â€“200 mA, supports 5V USB
- Dimensions: 65 mm Ã— 30 mm Ã— 5 mm
- Cost: ~â‚¹1,200 INR

Alternative ESP32-S3 with integrated DAC (~â‚¹800 INR) can perform TTS directly without separate Pi, but requires more complex C++ code.

Ultrasonic Sensors: HC-SR04
- Operating Voltage:5V
- Measurement Range: 2 cm â€“ 4 m (400 cm)
- Trigger Pulse: 10 Âµs minimum
- Echo Output: Pulse width proportional to distance
- Frequency:40 kHz
- Accuracy:Â±2% typical
- Quantity: 3 (front, left, right) â†’ â‚¹100 each

Mounting Configuration:
- Front sensor: Center-aligned at top of stick head (~15 cm above grip)
- Left sensor: 45Â° left of center
- Right sensor: 45Â° right of center
- Vertical spacing: ~5 cm between sensors to avoid beam cross-talk

---

Vibration Motor
- Type:Coin ERM (eccentric rotating mass)
- Operating Voltage: 3V (range 2â€“4V)
- Nominal Current: 100 mA
- Size: 10 mm diameter Ã— 3 mm thickness
- Frequency Response: 125â€“250 Hz (perceptible to skin)
- Cost: ~â‚¹100 INR

Driver Circuit:
- N-channel MOSFET (2N7002 or IRF520N)
- Gate resistor: 100 Î© (current limiting)
- Flyback diode: 1N4007 (back-EMF protection)
- Source to ground, drain to motor positive

Battery & Charging
- Primary Cell:18650 Li-ion, 3.7V nominal, 2600 mAh typical (runtime ~6â€“8 hours)
- Alternative: Two 18650 in parallel for double capacity (12â€“16 hours), or series for 7.4V with dual buck converters
- Charger: TP4056 (USB input, auto-cutoff at 4.2V, ~500 mA charge current)
- Protection:
  - Reverse polarity diode on battery input
  - Low-battery alert when cell voltage drops below 3.0V
- Cost: Battery ~â‚¹300, Charger board ~â‚¹50

Voltage Regulation
- Buck Converter: LM2596 (5V output, 3A capable)
  - Input: 3.7â€“7.4V (from battery or parallel cells)
  - Output: Stable 5V for ESP32, Raspberry Pi, sensors
  - Efficiency: ~80%
- **Cost:** ~â‚¹150

Mechanical & Enclosure
- Stick Material: Aluminum or PVC tube, 2.5â€“3 cm diameter, 80â€“100 cm length
- Sensor Housing: 3D-printed or machined plastic cap fitting stick head
- Handle Box: Small plastic enclosure (10 cm Ã— 8 cm Ã— 5 cm) for electronics, mounting on lower stick body
- Mounting Hardware: Zip ties, hot glue, small bolts/screws
- Weatherproofing: Silicone sealant around sensor ports, IP65-rated enclosure recommended

Connectors & Wiring
- Battery Connector:XT60 or JST (2-pin)
- Motor Connector 2-pin JST
- Sensor Connectors: 4-pin JST (Vcc, GND, Trig, Echo)
- Wiring Gauge: 22 AWG for signal, 20 AWG for power
- Cable Length:~1.5 m total to allow flexible routing in stick

---

4.2 Software Specification

ESP32 Firmware (Arduino IDE, C/C++)

Required Libraries:
- `Arduino.h` (core functionality)
- Hardware UART for serial communication (built-in)
- `driver/gpio.h`, `driver/ledc.h` for GPIO and PWM control (built-in)

Core Functions:
```
- setup()             : Initialize pins, UART, PWM
- loop()              : Main sensor reading cycle
- measureDistance()   : Read HC-SR04 via pulseIn()
- vibrationControl()  : Set PWM intensity based on threat level
- serialReport()      : Send TTS message to Raspberry Pi
- debounce()          : Filter sensor noise
- lowBatteryCheck()   : ADC monitoring
```

Key Specifications:

- Loop frequency: ~8 Hz per sensor (120 ms cycle)
- Serial baud: 115200
- PWM frequency: 5 kHz (smooth vibration)
- ADC sampling: Every 10 seconds for battery check

-Raspberry Pi Voice Script (Python 3)

Required Libraries:
- `pyserial` (serial port communication)
- `subprocess` (launch espeak TTS)
- `time` (delay and throttling)

Core Script (`pi_tts_listener.py`):
```python
- Read from ESP32 serial port
- Parse incoming messages (JSON-like format)
- Generate human-readable phrases
- Call espeak with spoken text
- Log events to file (optional)
- Handle connection errors gracefully
```

Key Specifications:
- Serial baud: 115200
- Minimum gap between TTS calls: 1.2 seconds
- Timeout on serial read: 1 second
- Restart on connection loss: Auto-reconnect every 5 seconds

Communication Protocol

Message Format (ESP32 â†' Raspberry Pi):
```
TTS,<LEVEL>,<DIRECTION>,<DISTANCE>
```

Examples:

- `TTS,VERY_CLOSE,front,25` â†' "Very close obstacle on the front, 25 centimeters."
- `TTS,NEAR,left,50` â†' "Obstacle 50 centimeters on the left."
- `TTS,FAR,right,150` â†' "Obstacle detected on the right."
- `MODE_PRESS` â†' "Mode changed." (toggle feedback)
- `BATTERY_LOW` â†' "Battery low, please recharge." (triggered at <3.0V)
- `SYSTEM_TEST_OK` â†' "All sensors functioning normally." (test button press)

Message Priority:
1. BATTERY_LOW (highest)
2. VERY_CLOSE
3. NEAR
4. FAR (lowest)
5. Mode/system messages

Configuration Parameters (in code, adjustable without reflashing)

```
VERY_CLOSE_THRESHOLD   = 30 cm
NEAR_THRESHOLD         = 80 cm
FAR_THRESHOLD          = 150 cm

VERY_CLOSE_VIBRATION   = 200 (0â€“255)
NEAR_VIBRATION         = 100
FAR_VIBRATION          = 40

MIN_TTS_GAP            = 1200 ms
```

SENSOR_READ_INTERVAL = 120 ms
BATTERY_CHECK_INTERVAL = 10000 ms
BATTERY_LOW_VOLTAGE = 3.0 V (nominal)

INDOOR_MODE = More sensitive, shorter thresholds
OUTDOOR_MODE = Less sensitive, longer thresholds
```

---

\newpage

5. Requirement Specification

## 5.1 Functional Requirements

FR1: Obstacle Detection
- The system SHALL detect static and moving obstacles within a range of 2 cm to 4 meters in front, left, and right directions.
- Detection accuracy SHALL be Â±5 cm for distances 30â€“200 cm.
- The system SHALL update detection at least every 200 ms.
- **Acceptance Criteria:** Obstacle positioned at 50 cm in front is detected and reported within 200 ms.

FR2: Directional Identification**
- The system SHALL identify and report obstacle direction as "front," "left," or "right" based on sensor array.
- For corner obstacles (equidistant from two sensors), the system SHALL report the predominant direction based on closer sensor.
- **Acceptance Criteria:** Obstacle at 45Â° angle reported as either "left-front" or "front-left"; left-most obstacle reported as "left."

FR3: Distance Quantization**
- The system SHALL report obstacle distance in three bands:
  - Very Close: <30 cm
  - Near: 30â€“80 cm
  - Far: 80â€“200 cm
- The system SHALL report exact distance (in cm) in voice alert for NEAR and FAR bands.
- Acceptance Criteria:** 45 cm obstacle generates "Obstacle 45 centimeters" alert, not just "Near."

FR4: Vibration Feedback**
- The system SHALL vary vibration intensity proportionally to proximity:
  - Very Close: PWM 200/255 (maximum intensity, continuous)
  - Near: PWM 60â€“150 (medium, pulsed)
  - Far: PWM 40 (light, brief pulse)
```

- Vibration response latency SHALL be <100 ms after detection.
- **Acceptance Criteria:** Hand on vibration motor feels strong vibration for Very Close, medium for Near, light for Far.

FR5: Voice Alert Generation**
- The system SHALL generate natural speech alerts using offline TTS (espeak on Raspberry Pi).
- Voice alert SHALL include direction and distance information.
- Voice alert SHALL not repeat more frequently than once per 1.2 seconds for same obstacle.
- Speech rate SHALL be 150â€“180 words per minute (normal conversational speed).
- **Acceptance Criteria:** Clear audible alert "Obstacle 50 centimeters on the left" within 1 second of detection.

FR6: Sensitivity Mode Switching**
- The system SHALL support two modes: **Indoor** and **Outdoor**.
  - **Indoor Mode:** Thresholds reduced by 20%; more frequent alerts; default for enclosed spaces.
  - Outdoor Mode:** Thresholds increased by 20%; fewer alerts; default for open areas.
- Mode change SHALL be triggered by user pressing the Mode button.
- Confirmation SHALL be provided via beep or brief voice message.
- **Acceptance Criteria:** Button press toggles mode; confirmation received within 500 ms.

FR7: Battery Monitoring
- The system SHALL continuously monitor battery voltage via ADC.
- When battery voltage drops below 3.0V nominal (cell discharge threshold):
  - System SHALL trigger low-battery alert ("Battery low, please recharge").
  - System SHALL reduce vibration intensity by 30% to extend runtime.
  - System SHALL reduce TTS frequency to once per 3 seconds.
- Acceptance Criteria:** Low-battery alert triggered at expected voltage; system continues operation for >30 min at reduced performance.

FR8: Power Management
- System SHALL enter low-power mode after 5 minutes of inactivity (no obstacles detected).
  - Sensor polling reduced to 2 Hz.
  - Wi-Fi/BLE disabled (if applicable).
  - Power consumption reduced to <50 mA.
- System SHALL resume full operation immediately upon detecting obstacle or button press.
- **Acceptance Criteria:** Multimeter measures <50 mA in idle mode; <100 mA in active mode.

FR9: User Interface Controls
- **Power Button:** Single press toggles system on/off. Held 3 seconds forces shutdown.
- **Mode Button:** Single press toggles Indoor/Outdoor mode; provides audio feedback.
- **Test Button:** Single press executes self-test; reports status via voice ("All sensors OK" or "Sensor fault").
- Each button SHALL be debounced with 50 ms delay.

- Acceptance Criteria: Buttons respond consistently without chatter; mode toggle confirmed within 500 ms.

FR10: Error Handling & Robustness
- If any sensor fails to echo within 30 ms timeout, system SHALL mark that reading as invalid (distance = 999).
- If two of three sensors fail, system SHALL alert user and continue with remaining sensor.
- If serial link to Raspberry Pi drops, vibration motor SHALL continue functioning independently.
- System SHALL automatically reconnect Pi on serial recovery.
- **Acceptance Criteria:** Unplugged sensor gracefully degraded; voice alerts resume when Pi reconnected.

5.2 Non-Functional Requirements

NFR1: Performance & Latency
- End-to-end latency (obstacle entry to user alert) SHALL be <500 ms.
- Sensor reading cycle time SHALL be <150 ms per complete 3-sensor sweep.
- Vibration response time SHALL be <100 ms.
- Voice alert generation + playback SHALL complete within 2 seconds.
- **Metric:** Oscilloscope measurement from obstacle entry to vibration onset.

NFR2: Reliability & Availability
- System mean time between failures (MTBF) SHALL exceed 200 hours.
- System SHALL gracefully degrade if one sensor fails (continue with two).
- System availability target: 95% (over 100 hours of cumulative use).
- **Metric:** Field testing over 200+ hours; log failure events.

NFR3: Power Consumption
- Active mode (all sensors reading): <150 mA @ 5V
- Idle mode (reduced polling): <50 mA @ 5V
- Sleep mode (power button off): <5 mA (due to control circuit leakage)
- Runtime on single 18650 (2600 mAh) battery: >6 hours continuous active use.
- **Metric:** Battery drain test; calculate hours from mAh and average current.

NFR4: Usability & Accessibility
- Device SHALL be operable with one hand (all buttons reachable).
- Device weight SHALL not exceed 500 g (including battery and enclosure).
- Device SHALL be mountable on standard canes (26â€“32 mm diameter).
- Voice feedback SHALL use clear, natural pronunciation at adjustable volume (0â€“100%).
- **Metric:** User acceptance testing; blind volunteer feedback.

NFR5: Durability & Environmental
- Enclosure SHALL be rated IP54 minimum (splash-resistant).

- Operating temperature range: 0Â°C to 50Â°C.
- Device SHALL survive drops from 1 meter onto hard floor without functional damage.
- Sensors SHALL remain calibrated after 500 hours of operation (Â±5% drift).
- **Metric:** Water splash test, temperature chamber cycling, drop test.

NFR6: Cost Efficiency
- Bill of materials (BOM) cost SHALL NOT exceed â‚¹3,500 INR per unit.
- Breakeven production volume: <500 units (justifies development cost).
- Cost per additional unit (mass production): <â‚¹2,500 INR.
- **Metric:** BOM review; supplier quotes.

NFR7: Security & Privacy
- System SHALL operate offline (no cloud connectivity required).
- Voice alerts SHALL not transmit personal location or user identity data.
- Battery charge/discharge data SHALL not be logged or transmitted externally.
- **Metric:** Network traffic capture; code review for data exfiltration.

NFR8: Maintainability & Extensibility
- Source code SHALL be documented with comments explaining all non-trivial logic.
- Firmware SHALL be open-source (MIT or Apache 2.0 license).
- Hardware shall be reverse-engineerable from schematics (no proprietary components).
- Code SHALL support easy modification of thresholds and parameters without recompilation (via config file on Pi).
- **Metric:** Code review; documentation completeness; ease of parameter update.

NFR9: Testability
- Firmware SHALL include built-in self-test (triggered by Test button).
- Self-test SHALL verify all GPIO outputs and sensor connectivity.
- Test results SHALL be reported via voice or serial console.
- Automated test suite SHALL cover >80% code paths.
- **Metric:** Code coverage report; successful self-test execution.

NFR10: Scalability & Future-Proofing
- Architecture SHALL support future integration of GPS, Bluetooth, or camera modules.
- Code modularity SHALL allow easy addition of new feedback mechanisms (e.g., additional buzzers, haptic gloves).
- System SHALL be upgradeable with firmware OTA (Over-The-Air) updates (if Bluetooth added later).
- **Metric:** Modular codebase structure; GPIO expansion port availability.

6. Feasibility Analysis

6.1 Technical Feasibility

Component Availability:
âœ… **Highly Feasible** â€” All core components (ESP32, Raspberry Pi, HC-SR04, 18650 batteries, MOSFET) are widely available from major suppliers (Adafruit, AliExpress, local electronics shops). Lead time: 2â€“4 weeks. Standard supply chain, no custom fabrication required.

Integration Complexity:
âœ… **Feasible** â€” The architecture is straightforward:
- UART serial communication is a well-established standard.
- Ultrasonic sensor interfacing is thoroughly documented (Arduino libraries available).
- PWM vibration control is a basic GPIO operation.
- TTS via espeak is mature Linux software.
- No complex protocols or custom hardware required.

Development Timeline:
- Hardware assembly & breadboarding: 1 week
- Firmware development & testing: 2 weeks
- Raspberry Pi script & TTS integration: 1 week
- Enclosure design & mounting: 1 week
- System integration & debugging: 1 week
- Field testing & documentation: 2 weeks
- **Total: ~8â€“10 weeks for a full project cycle (reasonable for semester project)**

Skill Requirements:
- Embedded C/C++ programming: Intermediate (Arduino/ESP32)
- Python scripting: Basic
- PCB soldering: Basicâ€“Intermediate
- Linux command line: Basic (for Raspberry Pi setup)
- Electronics troubleshooting: Intermediate
- **Assessment:** Within reach of engineering undergraduates with prior experience in one or more embedded systems courses.

6.2 Economic Feasibility

Bill of Materials (Estimated Cost in INR):

Item	Qty	Unit Cost	Total
ESP32 DevKit	1	â‚¹600	â‚¹600
Raspberry Pi Zero W	1	â‚¹1,200	â‚¹1,200
HC-SR04 Sensor	3	â‚¹100	â‚¹300

Coin Vibration Motor	1	â‚¹100	â‚¹100
MOSFET (2N7002)	1	â‚¹20	â‚¹20
Diode (1N4007)	1	â‚¹5	â‚¹5
18650 Battery + Holder	1	â‚¹300	â‚¹300
TP4056 Charger Board	1	â‚¹50	â‚¹50
LM2596 Buck Converter	1	â‚¹150	â‚¹150
Capacitors, Resistors, PCB	-	-	â‚¹100
Wiring, Connectors, Solder	-	-	â‚¹80
Enclosure, Mounting Hardware	-	-	â‚¹250
Total BOM			**â‚¹3,155**

âœ… **Under â‚¹3,500 target** â€" Feasible within typical student project budget.

Cost Reduction Opportunities (for mass production):
- Bulk purchasing of sensors: â‚¹50â€“60 each (vs. â‚¹100)
- Custom PCB: Saves â‚¹80 (wiring/breadboard cost)
- Integrated power module (single chip): Saves â‚¹100
- **Projected cost at 500-unit production: â‚¹2,200â€“â‚¹2,500 per unit** â€" still commercially viable.

Funding & Sponsorship:
- University maker labs typically provide soldering equipment, breadboards, and basic components.
- Raspberry Pi and ESP32 often available in ECE department inventory.
- **Estimated out-of-pocket cost per student: â‚¹1,500â€“â‚¹2,000** (acceptable for a semester project).

6.3 Operational Feasibility

User Training & Adoption:
âœ… Highly Feasible â " The interface is intuitive:
- Simple button controls (power, mode, test)
- Familiar metaphor (cane-mounted assistive device)
- No app or complex configuration required
- Initial orientation: 15â€“30 minutes

Maintenance & Support:
âœ… **Feasible** â€" Routine maintenance is minimal:
- Battery replacement/recharging: Standard procedure (like any rechargeable device)
- Firmware updates: Simple USB upload (if user has computer access)
- Sensor calibration: Manual if needed (measure and verify range)
- Expected device lifespan: 3â€“5 years with normal use

Field Deployment:
âœ… **Feasible** â€” Suitable for:
- Individual visually impaired users
- Rehabilitation centers & training facilities
- University disability services
- Community health programs

Scalability:
âœ… **Scalable** â€” Once prototype is validated:
- Firmware can be cloned across devices
- Documentation supports DIY replication
- Manufacturing partnership feasible for volume production

6.4 Risk Analysis

Risk	Likelihood	Impact	Mitigation
Sensor echo timeout (>4m range)	Medium	Low	Implement timeout; log failed reads; test in open field.
Vibration motor burn-out	Low	Medium	Add PWM frequency limiting; test motor under continuous load.
Raspberry Pi connectivity issues	Medium	Medium	Redundant operation (vibration works standalone); auto-reconnect logic.
False positives in noisy environment	Medium	Medium	Implement sensor fusion & hysteresis; add confidence threshold.
Battery voltage droop under peak load	Low	Low	Add capacitor across supply; test with all devices active.
Enclosure waterproofing failure	Low	Medium	Use IP65-rated enclosure; test with water spray; regular inspection.
Ultrasonic crosstalk (three sensors)	Low	Low	Increase sensor spacing; adjust trigger timing; stagger reads.
User discomfort with vibration intensity	Medium	Low	Make PWM intensity adjustable; offer multiple motor options.

---

9. Coding

This section contains the complete firmware and scripts for the Smart Blind Stick system.

9.1 ESP32 Firmware (Arduino Sketch)

File: smart_blind_stick_esp32.ino

```
 * Smart Blind Stick - ESP32 Firmware
 *
 * Purpose: Read three HC-SR04 ultrasonic sensors, assess threats,
 * control vibration motor with PWM, and send TTS messages to Raspberry Pi.
 *
 * Hardware:
 * - ESP32 DevKit
 * - 3x HC-SR04 ultrasonic sensors (front, left, right)
 * - 1x Coin vibration motor with MOSFET driver
 * - 3x Push buttons (power, mode, test)
 * - 18650 battery + power regulation
 *
 * Serial Communication: 115200 baud to Raspberry Pi
 *
 * Author: [Your Name]
 * Date: [Current Date]
 * Version: 1.0
 */

#include <Arduino.h>

//
================================================================================
=======
// PIN DEFINITIONS
//
================================================================================
=======

// HC-SR04 Sensor Pins
const int TRIG_FRONT = 18;
const int ECHO_FRONT = 19;
const int TRIG_LEFT  = 21;
const int ECHO_LEFT  = 22;
const int TRIG_RIGHT = 23;
const int ECHO_RIGHT = 25;

// Vibration Motor PWM
```

```
const int VIBRATION_PIN = 26;
const int PWM_CHANNEL = 0;
const int PWM_FREQUENCY = 5000;  // 5 kHz
const int PWM_RESOLUTION = 8;    // 8-bit (0-255)

// User Interface Buttons
const int MODE_BUTTON = 15;
const int TEST_BUTTON = 17;
const int POWER_BUTTON = 0;

// Battery ADC
const int BATTERY_PIN = 35;  // ADC pin for voltage monitoring

//
================================================================================
=======
// CONFIGURATION & THRESHOLDS
//
================================================================================
=======

// Distance thresholds (in cm) - adjustable for different environments
int VERY_CLOSE_THRESHOLD = 30;
int NEAR_THRESHOLD       = 80;
int FAR_THRESHOLD        = 150;

// Vibration intensity levels (PWM 0-255)
const int VIBRATION_VERY_CLOSE = 200;
const int VIBRATION_NEAR_MAX   = 150;
const int VIBRATION_NEAR_MIN   = 60;
const int VIBRATION_FAR        = 40;

// Timing thresholds (milliseconds)
const unsigned long MIN_TTS_GAP = 1200;      // Minimum gap between TTS messages
const unsigned long SENSOR_READ_INTERVAL = 120;
const unsigned long BATTERY_CHECK_INTERVAL = 10000;
const unsigned long DEBOUNCE_DELAY = 50;

// Sensor timeout (microseconds) - for pulseIn()
const unsigned long PULSE_TIMEOUT = 30000;  // 30 ms

// Battery monitoring
const float BATTERY_LOW_VOLTAGE = 3.0;  // Volts (nominal)
const int ADC_MAX = 4095;
const float ADC_REFERENCE = 3.3;
const float VOLTAGE_DIVIDER_RATIO = 2.0;  // If using voltage divider
```

```
//
================================================================================
=======
// GLOBAL VARIABLES
//
================================================================================
=======

// Timing
unsigned long lastSensorRead = 0;
unsigned long lastSpeak = 0;
unsigned long lastBatteryCheck = 0;

// Mode and state
bool indoorMode = true;  // true = Indoor, false = Outdoor
bool systemActive = true;

// Button states
bool lastModeButtonState = HIGH;
bool lastTestButtonState = HIGH;

// Sensor readings
long distanceFront = 999;
long distanceLeft = 999;
long distanceRight = 999;

//
================================================================================
=======
// SETUP: Initialize hardware and communication
//
================================================================================
=======

void setup() {
  // Serial communication with Raspberry Pi
  Serial.begin(115200);
  delay(500);
  Serial.println("=== Smart Blind Stick Initialized ===");

  // Configure sensor pins
  pinMode(TRIG_FRONT, OUTPUT);
  pinMode(ECHO_FRONT, INPUT);
  pinMode(TRIG_LEFT, OUTPUT);
  pinMode(ECHO_LEFT, INPUT);
```

```
  pinMode(TRIG_RIGHT, OUTPUT);
  pinMode(ECHO_RIGHT, INPUT);

  // Configure vibration motor PWM
  ledcSetup(PWM_CHANNEL, PWM_FREQUENCY, PWM_RESOLUTION);
  ledcAttachPin(VIBRATION_PIN, PWM_CHANNEL);
  ledcWrite(PWM_CHANNEL, 0);  // Start with no vibration

  // Configure button pins (pull-up)
  pinMode(MODE_BUTTON, INPUT_PULLUP);
  pinMode(TEST_BUTTON, INPUT_PULLUP);
  pinMode(POWER_BUTTON, INPUT_PULLUP);

  // Configure battery ADC
  pinMode(BATTERY_PIN, INPUT);
  analogSetAttenuation(ADC_11db);  // For 0-3.3V range

  // Startup beep via vibration (brief pulse)
  ledcWrite(PWM_CHANNEL, 100);
  delay(200);
  ledcWrite(PWM_CHANNEL, 0);

  // Send startup message
  Serial.println("SYSTEM_STARTUP");
}

//
================================================================================
=======
// HELPER FUNCTION: Measure distance from HC-SR04 sensor
//
================================================================================
=======

/**
 * Measures distance using HC-SR04 ultrasonic sensor.
 *
 * Principle:
 *   1. Send 10 Âµs trigger pulse to TRIG pin
 *   2. Measure duration of echo pulse on ECHO pin
 *   3. Distance = (duration in Âµs / 58) cm (or duration * 0.034 / 2)
 *
 * Returns: Distance in cm, or 999 if timeout or error
 */
long measureDistance(int trigPin, int echoPin) {
  // Send trigger pulse (10 Âµs)
```

```
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Measure echo pulse duration
  long duration = pulseIn(echoPin, HIGH, PULSE_TIMEOUT);

  // Check for timeout
  if (duration == 0) {
    return 999;  // Sensor not responding or obstacle too far
  }

  // Calculate distance in cm
  // Speed of sound: 343 m/s = 0.0343 cm/Âµs
  // Distance = time * speed / 2 (divide by 2 because sound travels to object and back)
  long distance = duration * 0.034 / 2;

  return distance;
}

//
================================================================================
=======
// HELPER FUNCTION: Debounce button input
//
================================================================================
=======

bool debounceButton(int pin, bool &lastState) {
  bool currentState = digitalRead(pin);

  if (currentState != lastState) {
    delay(DEBOUNCE_DELAY);
    currentState = digitalRead(pin);

    if (currentState != lastState) {
      lastState = currentState;
      return (currentState == LOW);  // Button pressed when LOW
    }
  }

  return false;
}
```

```
//
================================================================================
=======
// HELPER FUNCTION: Read battery voltage
//
================================================================================
=======

float readBatteryVoltage() {
  // Read ADC value
  int rawValue = analogRead(BATTERY_PIN);

  // Convert to voltage
  // If using voltage divider: V_actual = (ADC_reading / ADC_MAX) * ADC_REF *
DIVIDER_RATIO
  float voltage = (rawValue / (float)ADC_MAX) * ADC_REFERENCE *
VOLTAGE_DIVIDER_RATIO;

  return voltage;
}

//
================================================================================
=======
// HELPER FUNCTION: Threat assessment and vibration control
//
================================================================================
=======

void assessThreatAndVibrate(long minDistance, const char* side) {
  // Determine threat level and set vibration intensity
  if (minDistance < VERY_CLOSE_THRESHOLD) {
    // VERY CLOSE - imminent collision
    ledcWrite(PWM_CHANNEL, VIBRATION_VERY_CLOSE);

    if (millis() - lastSpeak > MIN_TTS_GAP) {
      Serial.printf("TTS,VERY_CLOSE,%s,%ld\n", side, minDistance);
      lastSpeak = millis();
    }
  }
  else if (minDistance < NEAR_THRESHOLD) {
    // NEAR - approaching hazard
    // Vibration intensity varies with distance (closer = stronger)
    int intensity = map(minDistance, VERY_CLOSE_THRESHOLD, NEAR_THRESHOLD,
                        VIBRATION_NEAR_MAX, VIBRATION_NEAR_MIN);
    ledcWrite(PWM_CHANNEL, intensity);
```

```
    if (millis() - lastSpeak > MIN_TTS_GAP) {
      Serial.printf("TTS,NEAR,%s,%ld\n", side, minDistance);
      lastSpeak = millis();
    }
  }
  else if (minDistance < FAR_THRESHOLD) {
    // FAR - awareness distance
    // Brief light pulse, less frequent TTS
    ledcWrite(PWM_CHANNEL, VIBRATION_FAR);

    if (millis() - lastSpeak > MIN_TTS_GAP * 2) {  // Less frequent TTS
      Serial.printf("TTS,FAR,%s,%ld\n", side, minDistance);
      lastSpeak = millis();
    }

    delay(100);  // Brief vibration pulse
    ledcWrite(PWM_CHANNEL, 0);
  }
  else {
    // FAR - no vibration
    ledcWrite(PWM_CHANNEL, 0);
  }
}

//
================================================================================
=======
// MAIN LOOP: Continuous operation
//
================================================================================
=======

void loop() {
  unsigned long currentTime = millis();

  //
================================================================================
===
  // BUTTON HANDLING
  //
================================================================================
===

  // Mode button - toggle Indoor/Outdoor sensitivity
  if (debounceButton(MODE_BUTTON, lastModeButtonState)) {
```

```
  indoorMode = !indoorMode;

  if (indoorMode) {
    VERY_CLOSE_THRESHOLD = 25;
    NEAR_THRESHOLD = 75;
    FAR_THRESHOLD = 140;
    Serial.println("MODE_INDOOR");
  } else {
    VERY_CLOSE_THRESHOLD = 35;
    NEAR_THRESHOLD = 85;
    FAR_THRESHOLD = 160;
    Serial.println("MODE_OUTDOOR");
  }

  // Haptic feedback: two short vibrations
  for (int i = 0; i < 2; i++) {
    ledcWrite(PWM_CHANNEL, 150);
    delay(100);
    ledcWrite(PWM_CHANNEL, 0);
    delay(100);
  }
}

// Test button - trigger self-test
if (debounceButton(TEST_BUTTON, lastTestButtonState)) {
  Serial.println("SYSTEM_TEST_START");

  // Test sensors
  long testF = measureDistance(TRIG_FRONT, ECHO_FRONT);
  long testL = measureDistance(TRIG_LEFT, ECHO_LEFT);
  long testR = measureDistance(TRIG_RIGHT, ECHO_RIGHT);

  // Log results
  Serial.printf("TEST_RESULTS: Front=%ld, Left=%ld, Right=%ld\n", testF, testL, testR);

  // Test vibration
  ledcWrite(PWM_CHANNEL, 200);
  delay(200);
  ledcWrite(PWM_CHANNEL, 0);

  Serial.println("SYSTEM_TEST_OK");
}

// Power button - force shutdown (optional)
if (digitalRead(POWER_BUTTON) == LOW) {
  delay(DEBOUNCE_DELAY);
```

```
  if (digitalRead(POWER_BUTTON) == LOW) {
    unsigned long holdTime = millis();
    while (digitalRead(POWER_BUTTON) == LOW) {
      delay(10);
      if (millis() - holdTime > 3000) {
        // Long press - shutdown
        ledcWrite(PWM_CHANNEL, 0);
        Serial.println("SYSTEM_SHUTDOWN");
        // In real device, cut power via relay or latch-off circuit
        while (true) delay(1000);  // Hang
      }
    }
  }
}

//
================================================================================
===
// SENSOR READING (every 120 ms)
//
================================================================================
===

if (currentTime - lastSensorRead >= SENSOR_READ_INTERVAL) {
  lastSensorRead = currentTime;

  // Read all three sensors
  distanceFront = measureDistance(TRIG_FRONT, ECHO_FRONT);
  distanceLeft = measureDistance(TRIG_LEFT, ECHO_LEFT);
  distanceRight = measureDistance(TRIG_RIGHT, ECHO_RIGHT);

  // Filter out invalid readings (999 = timeout)
  if (distanceFront == 999) distanceFront = 400;  // Assume far
  if (distanceLeft == 999) distanceLeft = 400;
  if (distanceRight == 999) distanceRight = 400;

  // Find minimum distance and determine direction
  long minDistance = min(distanceFront, min(distanceLeft, distanceRight));

  const char* side = "front";
  if (distanceLeft < distanceFront && distanceLeft <= distanceRight) {
    side = "left";
  } else if (distanceRight < distanceFront && distanceRight < distanceLeft) {
    side = "right";
  }
```

```
    // Debug output to Serial Monitor (optional)
    Serial.printf("SENSORS: F=%ld, L=%ld, R=%ld (Min=%ld, Side=%s)\n",
                  distanceFront, distanceLeft, distanceRight, minDistance, side);

    // Assess threat and control vibration
    if (minDistance < FAR_THRESHOLD) {
      assessThreatAndVibrate(minDistance, side);
    } else {
      ledcWrite(PWM_CHANNEL, 0);  // No vibration
    }
  }

  //
================================================================================
===
  // BATTERY MONITORING (every 10 seconds)
  //
================================================================================
===

  if (currentTime - lastBatteryCheck >= BATTERY_CHECK_INTERVAL) {
    lastBatteryCheck = currentTime;

    float batteryVoltage = readBatteryVoltage();

    if (batteryVoltage < BATTERY_LOW_VOLTAGE) {
      // Low battery alert
      Serial.println("BATTERY_LOW");

      // Reduce vibration intensity by 30%
      // (This is automatically handled in assessThreatAndVibrate if we modify logic)

      // Triple beep warning via vibration
      for (int i = 0; i < 3; i++) {
        ledcWrite(PWM_CHANNEL, 100);
        delay(100);
        ledcWrite(PWM_CHANNEL, 0);
        delay(100);
      }
    } else {
      // Battery status OK
      Serial.printf("BATTERY_OK: %.2f V\n", batteryVoltage);
    }
  }

  // Small delay to prevent overwhelming CPU
```

```
  delay(10);
}

//
================================================================================
=======
// OPTIONAL: Additional helper functions for advanced features
//
================================================================================
=======

/**
 * Advanced feature: Sensor fusion and edge detection
 * (Future enhancement to reduce false positives)
 */
void sensorFusionFilter() {
  // Could implement:
  // - Moving average filter for smoothing
  // - Hysteresis to avoid rapid on-off toggling
  // - Outlier rejection
}

/**
 * Advanced feature: Learning and calibration
 * (Future enhancement for user customization)
 */
void calibrateThresholds() {
  // Could allow user to teach the device:
  // - What "very close" feels like
  // - What "near" feels like
  // - Store calibration in EEPROM
}

//
================================================================================
=======
// END OF ESP32 FIRMWARE
//
================================================================================
=======
```

9.2 Raspberry Pi Voice Script (Python 3)

File: pi_tts_listener.py

```
#!/usr/bin/env python3
"""
Smart Blind Stick - Raspberry Pi TTS Listener

Purpose: Listen for serial messages from ESP32, parse them,
generate natural speech phrases, and play audio via espeak TTS.

Requirements:
  - pyserial (pip3 install pyserial)
  - espeak (apt install espeak)
  - Python 3.6+

Serial Protocol:
  Messages from ESP32 follow format: TTS,<LEVEL>,<DIRECTION>,<DISTANCE>
  Examples:
    TTS,VERY_CLOSE,front,25
    TTS,NEAR,left,50
    TTS,FAR,right,150
    MODE_PRESS
    BATTERY_LOW
    SYSTEM_STARTUP

Author: [Your Name]
Date: [Current Date]
Version: 1.0
"""

import serial
import subprocess
import time
import sys
import os
from datetime import datetime

#
================================================================================
=======
# CONFIGURATION
#
================================================================================
=======

SERIAL_PORT = '/dev/serial0'  # Standard Raspberry Pi UART
# Alternative if using USB adapter: '/dev/ttyUSB0'
BAUD_RATE = 115200
SERIAL_TIMEOUT = 1.0
```

```
# TTS Parameters
TTS_ENGINE = 'espeak'  # or 'pico2wave' for better quality
TTS_VOICE = 'english'
TTS_SPEED = 160  # words per minute (150-180 normal)
TTS_PITCH = 50  # 0-99
TTS_VOLUME = 100  # 0-100

# Audio output
AUDIO_OUTPUT = 'default'  # or 'alsa', 'pulse', etc.

# Logging
LOG_FILE = '/home/pi/blind_stick_log.txt'
ENABLE_LOGGING = True

# Rate limiting
MIN_MESSAGE_GAP = 1.2  # seconds between consecutive TTS

# ================================================================================
# HELPER FUNCTIONS
# ================================================================================

def log_message(message):
    """Log message to file with timestamp."""
    if ENABLE_LOGGING:
        try:
            with open(LOG_FILE, 'a') as f:
                timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                f.write(f"[{timestamp}] {message}\n")
        except Exception as e:
            print(f"[LOG ERROR] {e}")

def speak(text):
    """
    Generate and play speech using espeak TTS.

    Args:
        text (str): Text to speak

    Returns:
        bool: True if successful, False otherwise
    """
```

```
    try:
        # Build espeak command
        cmd = [
            TTS_ENGINE,
            '-v', TTS_VOICE,
            '-s', str(TTS_SPEED),
            '-p', str(TTS_PITCH),
            '-a', str(TTS_VOLUME),
            text
        ]

        print(f"[SPEAK] {text}")

        # Execute espeak (blocks until speech finishes)
        subprocess.run(cmd, check=True, capture_output=False)

        return True

    except subprocess.CalledProcessError as e:
        print(f"[TTS ERROR] espeak failed: {e}")
        log_message(f"TTS ERROR: {e}")
        return False

    except FileNotFoundError:
        print(f"[TTS ERROR] {TTS_ENGINE} not found. Install with: sudo apt install espeak")
        return False

def parse_message(message):
    """
    Parse serial message and extract components.

    Args:
        message (str): Serial message from ESP32

    Returns:
        dict: Parsed message components, or None if invalid
    """
    message = message.strip()

    if not message:
        return None

    if ',' in message:
        # TTS message format: TTS,<LEVEL>,<DIRECTION>,<DISTANCE>
        parts = message.split(',')
```

```
        if len(parts) >= 4 and parts[0] == 'TTS':
            return {
                'type': 'TTS',
                'level': parts[1].strip(),
                'direction': parts[2].strip(),
                'distance': parts[3].strip()
            }

    # Non-TTS messages (system events)
    return {
        'type': 'SYSTEM',
        'event': message
    }

def generate_phrase(msg_dict):
    """
    Generate natural speech phrase from parsed message.

    Args:
        msg_dict (dict): Parsed message from parse_message()

    Returns:
        str: Natural language phrase, or None if unrecognized
    """
    if msg_dict is None:
        return None

    if msg_dict['type'] == 'TTS':
        level = msg_dict['level']
        direction = msg_dict['direction']
        distance = msg_dict['distance']

        # Sanitize inputs
        direction = direction.lower()

        # Generate phrase based on threat level
        if level == 'VERY_CLOSE':
            phrase = f"Very close obstacle on the {direction}, {distance} centimeters."

        elif level == 'NEAR':
            phrase = f"Obstacle {distance} centimeters on the {direction}."

        elif level == 'FAR':
            phrase = f"Obstacle detected on the {direction}."

        else:
```

```
            phrase = f"Obstacle detected on the {direction}."

        return phrase

    elif msg_dict['type'] == 'SYSTEM':
        event = msg_dict['event']

        # Map system events to phrases
        event_phrases = {
            'MODE_INDOOR': "Mode changed to indoor.",
            'MODE_OUTDOOR': "Mode changed to outdoor.",
            'BATTERY_LOW': "Battery low. Please recharge.",
            'SYSTEM_STARTUP': "System started.",
            'SYSTEM_TEST_OK': "All sensors functioning normally.",
            'MODE_PRESS': "Mode changed.",
        }

        return event_phrases.get(event, f"System event: {event}")

    return None

#
================================================================================
=======
# MAIN LOOP
#
================================================================================
=======

def main():
    """Main listener loop."""

    print("=" * 60)
    print("Smart Blind Stick - Raspberry Pi TTS Listener")
    print("=" * 60)
    print(f"Serial Port: {SERIAL_PORT}")
    print(f"Baud Rate: {BAUD_RATE}")
    print(f"TTS Engine: {TTS_ENGINE}")
    print(f"Log File: {LOG_FILE}")
    print()

    # Open serial connection
    try:
        ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=SERIAL_TIMEOUT)
        print(f"[SUCCESS] Connected to {SERIAL_PORT}")
        log_message("Serial connection established")
```

```
    except serial.SerialException as e:
        print(f"[ERROR] Could not open {SERIAL_PORT}: {e}")
        print("Troubleshooting:")
        print("  1. Check if Raspberry Pi UART is enabled: sudo raspi-config -> Interface Options
-> Serial")
        print("  2. Verify wiring: ESP32 TX -> Pi RX (GPIO 14), RX -> TX (GPIO 15)")
        print("  3. Check permissions: sudo usermod -a -G dialout pi")
        sys.exit(1)

    last_tts_time = 0

    try:
        while True:
            # Read line from serial port
            if ser.in_waiting:
                try:
                    line = ser.readline().decode('utf-8', errors='ignore').strip()

                    if not line:
                        continue

                    print(f"[SERIAL] {line}")
                    log_message(f"RX: {line}")

                    # Parse message
                    parsed = parse_message(line)

                    if parsed is None:
                        continue

                    # Rate limit TTS messages
                    current_time = time.time()
                    if current_time - last_tts_time < MIN_MESSAGE_GAP:
                        print(f"[RATE LIMIT] Message skipped (gap too small)")
                        continue

                    # Generate phrase
                    phrase = generate_phrase(parsed)

                    if phrase:
                        # Speak the phrase
                        speak(phrase)
                        last_tts_time = time.time()
                        log_message(f"TX: {phrase}")
                    else:
```

```
                print(f"[WARNING] Could not generate phrase for: {line}")

            except UnicodeDecodeError as e:
                print(f"[DECODE ERROR] {e}")

            except Exception as e:
                print(f"[ERROR] {e}")
                log_message(f"ERROR: {e}")

        time.sleep(0.1)  # Small delay to prevent CPU busy-wait

    except KeyboardInterrupt:
        print("\n[INFO] Shutting down...")
        log_message("Listener stopped (user interrupt)")

    except Exception as e:
        print(f"[FATAL ERROR] {e}")
        log_message(f"FATAL: {e}")

    finally:
        if ser.is_open:
            ser.close()
            print("[INFO] Serial port closed")

#
================================================================================
=======
# STARTUP
#
================================================================================
=======

if __name__ == '__main__':
    main()
```

9.3 Installation Instructions

ESP32 Setup:

1. Install Arduino IDE (if not already installed)
   ```bash
   # On Linux/Mac/Windows, download from https://www.arduino.cc/en/software
   ```

2. **Add ESP32 Board**
   - Open Arduino IDE â†' Preferences
   - Add to "Additional Boards Manager URLs":
`https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
   - Tools â†' Board Manager â†' Search "ESP32" â†' Install by Espressif

3. **Upload Firmware**
   - Copy `smart_blind_stick_esp32.ino` to Arduino IDE
   - Connect ESP32 via USB
   - Tools â†' Board â†' "ESP32 Dev Module"
   - Tools â†' Port â†' Select correct COM port
   - Sketch â†' Upload

Raspberry Pi Setup:

1. **Install Required Packages**
   ```bash
   sudo apt update
   sudo apt install espeak python3-serial -y
   ```

2. **Enable Serial Port (UART)**
   ```bash
   sudo raspi-config
   # Navigate to: Interface Options â†' Serial â†' Enable serial interface
   # DO NOT enable login shell over serial
   ```

3. **Install Python Script**
   ```bash
   mkdir ~/blind_stick
   cp pi_tts_listener.py ~/blind_stick/
   chmod +x ~/blind_stick/pi_tts_listener.py
   ```

4. **Run Script (Manual)**

```bash
cd ~/blind_stick
python3 pi_tts_listener.py
```

5. **Run at Boot (Optional)**
   - Create systemd service file
   ```bash
   sudo nano /etc/systemd/system/blind-stick.service
   ```

   Add content:
   ```ini
   [Unit]
   Description=Smart Blind Stick TTS Listener
   After=network.target

   [Service]
   Type=simple
   User=pi
   WorkingDirectory=/home/pi/blind_stick
   ExecStart=/usr/bin/python3 /home/pi/blind_stick/pi_tts_listener.py
   Restart=always
   RestartSec=5

   [Install]
   WantedBy=multi-user.target
   ```

   Enable:
   ```bash
   sudo systemctl enable blind-stick.service
   sudo systemctl start blind-stick.service
   # Check status:
   sudo systemctl status blind-stick.service
   ```

9.4 Testing the Code

Bench Testing (without physical device)

1. Test ESP32 Firmware Alone
   - Upload to ESP32
   - Open Serial Monitor (115200 baud)
   - Watch sensor readings print
   - Verify messages: `SENSORS: F=XXX, L=XXX, R=XXX`

2. Test Serial Communication
   - Connect ESP32 to Raspberry Pi via UART
   - Run Python script on Pi
   - See serial messages printed to console
   - Verify TTS messages: `[SERIAL] TTS,NEAR,left,50`

3. Simulate Sensor Input (Mock Testing)
   - Modify firmware to use fixed distances for testing
   - Verify threat levels and vibration intensities
   - Test mode switching logic

Hardware Testing

1. Single Sensor
   - Place HC-SR04 on breadboard
   - Upload basic sketch to read distance
   - Hold objects at various distances (10 cm, 50 cm, 100 cm)
   - Verify readings in Serial Monitor

2. Three Sensors
   - Add left and right sensors
   - Verify no crosstalk (sensors read independently)
   - Test staggered trigger timing

3. Vibration Motor
   - Apply PWM signal with varying intensity (0-255)
   - Feel vibration intensity changes
   - Verify flyback diode protects MOSFET

4. Full System Integration
   - All sensors, motor, Pi connected
   - Place obstacles and measure alerts
   - Test latency with stopwatch
   - Verify voice output clarity and speech rate

---

12. Future Recommendations

Phase 2 Enhancements:

1. LIDAR Integration (VL53L1X)
   - Replace HC-SR04 with VL53L1X for better accuracy (<5 cm error)
   - Supports up to 4 m range
   - More robust to ambient light and materials
   - Cost: ~â‚¹500 per sensor (higher, but justified for accuracy)

2. GPS & GIS Mapping
   - Add NEO-6M GPS module
   - Create crowdsourced hazard map of local environment
   - Share obstacle locations with community
   - Enable navigation to specific destinations

3. Bluetooth Connectivity
   - Add Bluetooth module (HC-05) to ESP32
   - Allow smartphone app for real-time monitoring
   - Enable over-the-air firmware updates
   - Log trip history and statistics

4. Computer Vision
   - Integrate Raspberry Pi camera + TensorFlow Lite
   - Detect and classify obstacles (curbs, steps, people)
   - Read signs and text-to-speech
   - Recognize landmarks for navigation aid

5. Machine Learning
   - Train model on user's gait and typical movements
   - Predict intended path; proactive alerts
   - Adaptive sensitivity based on user's learning curve
   - Anomaly detection for unexpected obstacles

6. Extended Feedback
   - Haptic glove with multiple vibration motors
   - Bone conduction speaker for directional audio
   - Head-mounted display (if sighted family member assisting)
   - Augmented reality navigation overlay

7. Multi-User Support
   - Wireless sync between multiple users
   - Shared experience (friend guidance via smartphone)
   - Emergency SOS to caregivers with location

8. Environmental Adaptation

- Auto-calibration for temperature changes
- Adaptive gain for different obstacle types
- Learning outdoor noise reduction
- Dynamic threshold adjustment based on time of day

9. Industrial Applications
- Warehouse inventory robot
- Industrial safety alert system
- Obstacle detection for self-driving vehicles
- Autonomous wheelchairs and mobility devices

10. Commercial Viability
- Develop manufacturing partnership
- Target market: Rehab centers, NGOs for disabled
- Pricing: â‚¹2,500â€“â‚¹3,500 per unit (mass production)
- Warranty & after-sales support model

13. Bibliography

[1] World Health Organization (2021). "World Report on Vision." WHO Technical Reports Series 928.

[2] Legge, G. E., & Geruschat, D. R. (2016). "People with low vision." In Lighthouse International, Vision rehabilitation and low vision. Oxford University Press, pp. 109â€“128.

[3] Arduino Official Documentation. "HC-SR04 Ultrasonic Sensor." https://www.arduino.cc/en/Guide/HomePage

[4] Espressif Systems. "ESP32 Technical Reference Manual." https://www.espressif.com/en/products/microcontrollers/esp32/resources

[5] Raspberry Pi Foundation. "Raspberry Pi Zero W Datasheet." https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[6] Linux man page (7th ed.). "espeak - Text-to-Speech synthesizer." https://linux.die.net/man/1/espeak

[7] Kuc, R. (2000). "Introduction to Robotics and Intelligent Systems." Upper Saddle River: Prentice Hall.

[8] Gerchuk, B., & Brabyn, J. (2010). "Ultrasonic-based navigation aids for the blind: An overview." Journal of Blindness Innovation and Research, 3(1), 15â€“27.

[9] pySerial Documentation. Python Serial Port Library. https://pyserial.readthedocs.io/

[10] Texas Instruments. "LM2596 Datasheet: Simple Switcher Power Converter." https://www.ti.com/product/LM2596

[11] Infineon Technologies. "2N7002 N-Channel MOSFET." Datasheet.

[12] Panasonic. "18650 Li-Ion Battery Technical Specifications." Product Documentation.

[13] Anderson, R. H. (2015). "Assistive Technology for the Visually Impaired and Blind." Survey of Rehabilitation Engineering Research, Cambridge University Press.

[14] Schenkman, B. N., & JÃ¶nsson, E. (2000). "Phantom vibration syndrome among Swedish mobile phone users." Computers in Human Behavior, 16(6), 571â€“580.

[15] Molnar, A., & Csepregi, A. (2009). "Haptic feedback in virtual environments." International Journal of Advanced Media and Communication, 3(2), 112â€“128.