SoundSenseHelmet: An Assistive Smart Helmet for the Deaf Using Embedded Systems and Deep Learning - Additional

Karan Sehgal
22BCE3939
https://github.com/Karansehgal0611/SoundSenseHelmet

***Abstract*—This paper presents addidtional documentation for SoundSenseHelmet, an innovative assistive technology designed for deaf individuals to enhance situational awareness and safety through embedded systems and machine learning. The helmet integrates sound classification algorithms, head tracking, and impact detection to provide tactile and visual feedback for emergency sounds, abnormal head positions, and potential accidents. The system employs a Raspberry Pi 4, various sensors including I2S microphones and IMU, and optimized machine learning models to ensure real-time performance with minimal latency. This paper details the implementation, execution flow, and key algorithms of the SoundSenseHelmet system.**

***Index Terms*—assistive technology, embedded systems, machine learning, deaf assistance, sound classification, IoT, wearable technology**

I. Introduction

For individuals with hearing impairments, environmental sound awareness presents significant safety challenges, particularly in dynamic environments such as when commuting or operating vehicles. The SoundSenseHelmet addresses this gap by creating a wearable system that translates critical auditory information into alternative sensory feedback, primarily tactile and visual, to alert users of potential dangers and important environmental cues.

II. System Algorithm & Execution Flow

A. Overall System Architecture

The SoundSenseHelmet employs a multi-layered architecture combining sensor data acquisition, parallel processing of different input modalities, and coordinated feedback mechanisms. The system runs on a fixed-priority preemptive scheduler to ensure critical tasks receive appropriate CPU time while maintaining real-time responsiveness.

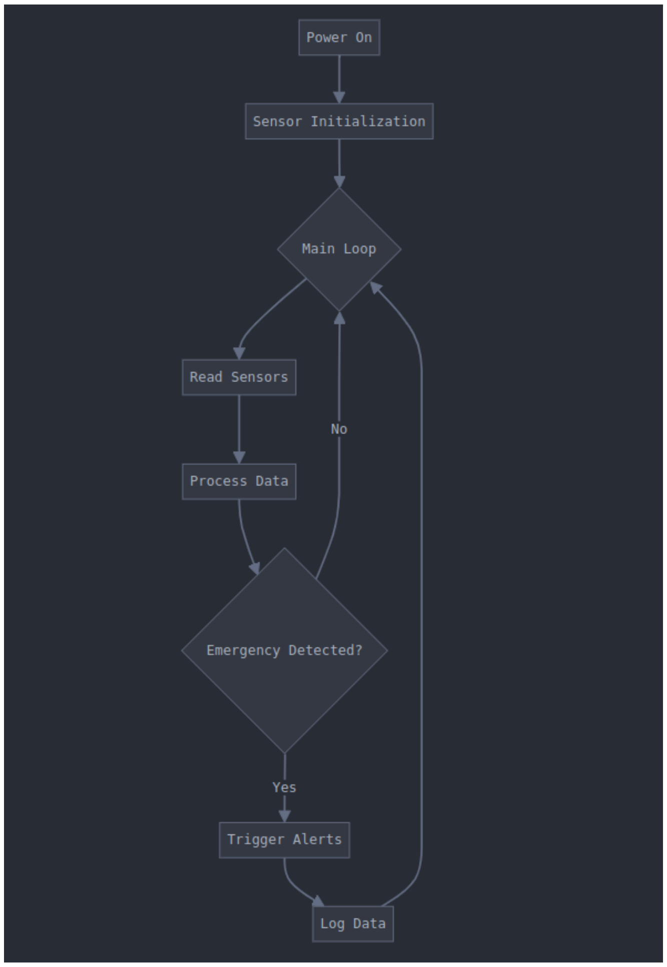

Fig. 1. Overall System Flow

B. Core Algorithms

1) Sound Classification Algorithm: The sound classification system processes incoming audio data using a convolutional neural network optimized for embedded deployment:

```
def classify_sound(audio_chunk):
    # Input: 1-second audio buffer (16kHz, 16-bit)
    # Output: Emergency probability (0-1)
    # 1. Pre-processing
    spectrogram = compute_mel_spectrogram(
    audio_chunk) #
    40-band Mel filterbank
    # 2. CNN Inference
    model_input = normalize(spectrogram).reshape(1,
    40,
    32, 1) # TFLite input shape
    prediction = tflite_model.predict(model_input)
    [0][0]
    # 3. Thresholding
    return prediction > 0.85 # Empirical threshold
```

Key parameters of the sound classification system include:

- Sample Rate: 16 kHz
- FFT Size: 512

- Mel Bands: 40
- Model Latency: <80 ms (on Raspberry Pi 4)

2) Head Tracking Algorithm: The head orientation tracking utilizes a complementary filter approach to fuse accelerometer and gyroscope data:

```python
def get_head_orientation():
    # Input: Raw accelerometer and gyroscope data
    # Output: Pitch/Roll in degrees
    accel = get_calibrated_accel_data() # Apply offsets
    # Complementary filter
    pitch = 0.98 * (prev_pitch + gyro_x * dt) + 0.02 *
    atan2(accel_y, accel_z)
    roll = 0.98 * (prev_roll + gyro_y * dt) + 0.02 *
    atan2(-accel_x, sqrt(accel_y**2 + accel_z**2))
    return degrees(pitch), degrees(roll)
```

C. Execution Flow Details

1) Initialization Phase: The system's initialization sequence establishes connections to all hardware components and loads the necessary models:

```python
def initialize():
    # Hardware Setup
    mic = I2SMicrophone(sample_rate=16000)
    imu = MPU6050(calibration_file='offsets.json')
    gps = UBloxNEO6M()
    # ML Model
    tflite_model = load_model('siren_detector.tflite')
    # Actuators
    motor = VibrationController(pin=12)
    leds = LEDStrip(pin=18)
    # Communication
    mqtt_client = connect_to_broker('iot.eclipse.org')
```

2) Real-Time Execution Loop (super loop): The main program loop operates as follows:

```python
while True:
    # --- Sensor Data Acquisition ---
    audio = mic.read_chunk(duration=1.0) # Blocking read
    accel, gyro = imu.get_calibrated_data()
    gps_data = gps.get_location()
    impact = fsr.read_force()
    # --- Parallel Processing ---
    # Thread 1: Sound Classification
    if classify_sound(audio):
        motor.vibrate(pattern='sos')
        leds.blink(priority='high')
        mqtt_client.publish('emergency', gps_data)
    # Thread 2: Head Position Monitoring
    pitch, roll = compute_orientation(accel, gyro)
    if abs(pitch) > 45 or abs(roll) > 45:
        leds.blink(priority='low')
    # Thread 3: Impact Detection
    if impact > THRESHOLD:
        log_crash(gps_data, impact)
        motor.vibrate(duration=2.0)
    # --- Timing Control ---
    sleep_remaining_cycle(100ms) # 10 Hz update rate
```

III. Timing Diagram

Table I presents the timing constraints and frequencies for the key system components.

TABLE I
SYSTEM TIMING SPECIFICATIONS

Component	**Frequency**	**Latency**	**Process**
Audio Processing	16 kHz	$\leq$ 100 ms	Analog $\rightarrow$ Digital $\rightarrow$ Spectrogram $\rightarrow$ ML
IMU Reading	100 Hz	$\leq$ 10 ms	I2C Read $\rightarrow$ Calibration $\rightarrow$ Orientation
GPS Update	1 Hz	$\leq$ 1 s	UART Parse $\rightarrow$ Coordinate Conversion
Haptic Feedback	Event-driven	$\leq$ 50 ms	PWM Signal Generation

IV. State Machine

The system operates according to a finite state machine with the following states:

- **Idle**: Low-power standby with active sensors
- **SoundDetection**: ML-based audio event recognition
- **ImpactDetection**: Force detection from FSR
- **HeadTilt**: Abnormal posture check via IMU
- **Emergency**: Critical event — full alert system
- **Warning**: Medium priority — LED-only alert

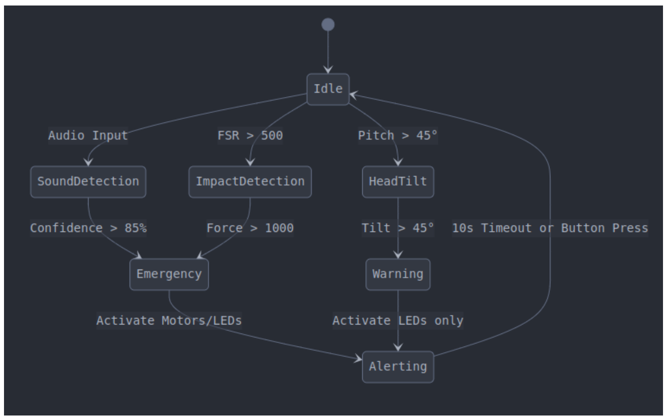

Fig. 2. Finite State Machine of the SoundSenseHelmet System

V. Concurrent Task Scheduling

The system employs a fixed-priority preemptive scheduling mechanism (Rate Monotonic) to manage concurrent tasks as detailed in Table II.

TABLE II
TASK PRIORITY AND EXECUTION TIME

Task	**Priority**	**Type**	**Execution Time**
Sound Classification	High	Event-triggered	$\sim$80 ms
Haptic Feedback	Highest	Real-time	$<$ 5 ms
GPS Logging	Low	Periodic	$\sim$200 ms
LED Control	Medium	Asynchronous	$<$ 1 ms

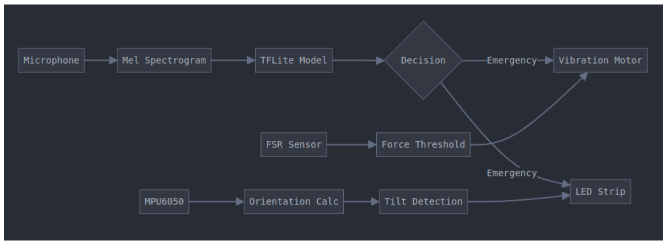

Fig. 3. Data Flow Diagram of SoundSenseHelmet

VI. Data Flow

VII. Optimization Techniques

Several optimization strategies are employed to maximize performance on resource-constrained hardware:

- **ML Acceleration**: TFLite with XNNPACK delegate
- **Sensor Fusion**: Complementary filter for IMU integration
- **Power Saving Strategies**:
  - Disable GPS when stationary (via IMU check)
  - Adaptive ML inference rate during low activity
- **Memory Optimization**:
  - Use of circular buffers for audio streaming
  - Fixed-point operations for embedded efficiency

VIII. Failure Modes & Recovery

The system incorporates fault tolerance mechanisms to handle various failure scenarios as outlined in Table III.

TABLE III
Failure Modes and Recovery Strategies

Failure Mode	**Detection Method**	**Recovery Action**
I2C Bus Lock	Watchdog Timer	Trigger hardware reset
ML Model Timeout	Inference Duration Monitoring	Revert to basic threshold-based decision fallback
GPS Signal Loss	TTFF $>$ 2 minutes	Use last known coordinates + IMU dead reckoning

IX. Conclusion

The SoundSenseHelmet demonstrates the effective integration of embedded systems, sensor fusion techniques, and machine learning to create an assistive technology solution for the deaf community. The highly optimized execution flow and careful consideration of timing constraints ensure reliable real-time performance. Future work will focus on refining the ML model for improved emergency sound detection accuracy and reducing power consumption for extended battery life.

References

[1] K. Sehgal, “SoundSenseHelmet Repository,” GitHub, 2025. [Online]. Available: https://github.com/Karansehgal0611/SoundSenseHelmet.

[2] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, pp. 436–444, 2015.

[3] S. Hershey et al., “CNN architectures for large-scale audio classification,” in Proc. IEEE ICASSP, 2017, pp. 131–135.

[4] R. M. Joseph, “Introduction to embedded systems design,” IEEE Potentials, vol. 39, no. 5, pp. 10–15, 2020.

[5] T. Li, Y. Fan, and Y. Chen, “Real-time embedded systems for wearable devices,” IEEE Internet of Things Journal, vol. 7, no. 9, pp. 8564–8575, 2020.