METHODOLOGY STEPS - MRITYUNJAY GUPTA

Development of an Embedded Speech-to-Text and Emotion Recognition Device with Haptic Feedback for Individuals with Hearing Impairments

I - Stepwise Methodology Explained in Day 1 as Introduction

This methodology integrates signal processing, machine learning, and embedded systems to create a wearable device that improves communication for individuals with hearing impairments. Each step has been detailed to ensure the reproducibility and optimization of the system.

1. **Understanding Libraries**: Imagine you have a magical toolbox. We pick special tools (libraries like `numpy`, `librosa`, etc.) to help us work on speech, emotions, and vibrations.
2. **Learning from Books**: Before starting, we read about others' work to learn the best ways to turn speech into text, understand emotions from voice, and create fun vibration signals.
3. **Collecting Sounds**: We record people talking in different moods (like happy or sad) and store these sounds to teach our system.
4. **Finding Patterns in Sounds**: Just like how a song has rhythm, every voice has hidden patterns. We find these patterns (features like pitch or energy) to understand the emotions behind the voice.
5. **Teaching a Robot to Feel**: Using the patterns we found, we teach a robot (model) how to guess if someone is happy, sad, or angry.
6. **Turning Voice into Words**: With smart tools (like Vosk), we teach our system to listen and write what it hears, just like a little helper.
7. **Creating Vibrations**: For each emotion, we create unique vibrations, like a secret handshake, to let someone "feel" the mood.
8. **Building a Tiny Computer**: We use a mini-computer (like Raspberry Pi) that can do all this work while being small enough to carry.
9. **Saving Battery**: We make sure our tiny computer doesn’t run out of energy quickly by putting it to sleep when not needed.
10. **Testing the Magic**: Finally, we check if everything works—if the system can hear, understand feelings, and vibrate correctly.

II - Stepwise Methodology for a Research Perspective

1. Importing and Preparing Libraries

Objective: Set up a modular environment to seamlessly transition between different stages of the research.

To begin, we use specialized libraries for signal processing, machine learning, and embedded system integration. For example:

- **`numpy`**: For numerical computations and handling arrays (e.g., audio features like MFCCs).
- **`librosa`**: For extracting audio features like pitch and MFCCs.
- **`tensorflow`**: For creating and training deep learning models.
- **`sklearn`**: For classification models such as SVM.

Code Used in Google Colab to achieve this Objective

```
import numpy as np
import librosa
import tensorflow as tf
from sklearn.svm import SVC
```

2. Literature Review

Objective: Identify gaps in current approaches and establish a foundation for innovation.

This step is critical to identify existing solutions and their limitations:

- **Speech-to-text**: Investigate lightweight models like Vosk or DeepSpeech and their adaptability to embedded platforms.
- **Emotion recognition**: Understand key features (MFCCs, pitch, prosody) and their role in emotion classification.

- **Haptic feedback**: Study mechanisms like vibrotactile motors and their applications in sensory substitution.

3. Data Collection and Preprocessing

Objective: Create a diverse and high-quality dataset for training and testing.

- **Audio Collection**: Gather audio datasets labeled with emotions (e.g., RAVDESS or custom recordings).
- **Preprocessing**:
    - Normalize audio for consistent volume.
    - Segment audio into fixed lengths for processing.
    - Augment data (e.g., add noise, pitch shifts) to make the model robust.

Code Used in Google Colab to achieve this Objective

```
audio, sr = librosa.load("path/to/audio.wav", sr=None)
mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13)
```

4. Feature Extraction

Objective: Represent audio in a format optimized for machine learning.

Extract meaningful features from the audio, which act as inputs to the machine learning model:

- **MFCCs**: Capture the timbre of the speech.
- **Pitch**: Provides cues to emotional intensity.
- **Energy**: Indicates loudness, a key emotional marker.

Code Used in Google Colab to achieve this Objective

```
pitch, _ = librosa.pyin(audio, fmin=50, fmax=300, sr=sr)
energy = librosa.feature.rms(y=audio)
features = np.concatenate((mfccs.mean(axis=1), [np.mean(pitch)],
[np.mean(energy)]))
```

5. Building the Emotion Recognition Model

Objective: Develop a robust, real-time classifier for emotions.

- Train a lightweight machine learning model:
    - **SVM**: Suitable for small datasets and interpretable decision boundaries.
    - **CNN**: Ideal for capturing spatial patterns in MFCC images.
- Use a train-test split to evaluate performance.

Code Used in Google Colab to achieve this Objective

```
clf = SVC(kernel="linear", probability=True)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
```

6. Speech-to-Text Conversion

Objective: Achieve efficient speech-to-text conversion on resource-constrained platforms.

- Use pre-trained models like **Vosk** (optimized for embedded systems) for transcription.
- Convert audio inputs into text, maintaining accuracy under noisy conditions.

Code Used in Google Colab to achieve this Objective

```
from vosk import Model, KaldiRecognizer

model = Model("path/to/vosk-model")
recognizer = KaldiRecognizer(model, sr)

if recognizer.AcceptWaveform(audio):
    result = recognizer.Result()
    print(result)
```

7. Designing the Haptic Feedback System

Objective: Translate auditory information into tactile feedback for sensory substitution.

- Develop vibration patterns to represent specific emotions or speakers:
  - Use vibrotactile motors to create custom feedback.
  - Design short bursts, long vibrations, or oscillating patterns for distinctiveness.
- Map these patterns to emotions or identities.

Code Used in Google Colab to achieve this Objective

```
def generate_haptic_feedback(emotion):
    patterns = {"happy": [0.5, 0.2], "sad": [1.0, 0.5], "angry": [0.2, 0.2, 0.2]}
    return patterns.get(emotion, [0.5])

def trigger_haptic(pattern):
    for duration in pattern:
        print(f"Vibrating for {duration} seconds")
        time.sleep(duration)
```

8. Embedded Hardware Selection

Objective: Minimize power consumption and latency while ensuring scalability.

- Choose the best platform (Raspberry Pi) based on computational needs.
- Convert models to lightweight formats like **TensorFlow Lite**.
- Optimize hardware-software integration for real-time performance.

9. Power Management

Objective: Extend battery life while maintaining functionality.

For wearable applications, efficient power usage is critical:

- Implement **low-power modes** to conserve energy when idle.
- Use hardware features like power gating to reduce consumption.

Code Used in Google Colab to achieve this Objective

```
def manage_power(mode):
    if mode == "low-power":
        print("Activating low-power mode...")
    elif mode == "performance":
        print("Activating performance mode...")
```

10. Testing and Validation

Objective: Ensure reliability, robustness, and user satisfaction.

- Conduct rigorous testing for each module:
  - Speech-to-text: Measure word error rate.
  - Emotion recognition: Evaluate accuracy, precision, recall.
  - Haptic feedback: Validate usability with user feedback.
- Integrate all components and test the complete system.

Code Used in Google Colab to achieve this Objective

```
text_result = recognizer.Result()
emotion_result = clf.predict(features.reshape(1, -1))
trigger_haptic(generate_haptic_feedback(emotion_result[0]))
```
