SINGSENSE: A VIRTUAL REALITY SINGING APP TO ALLEVIATE STRESS

Shengtian Hong [1], Yu Sun [2]

[1] Milton Academy, 170 Centre Street, Milton, MA02186
[2] California State Polytechnic University, Pomona, CA, 91768, Irvine, CA 92620

ABSTRACT

SingSense addresses the gap in traditional VR singing applications by integrating real-time biometric feedback to create an entertaining and immersive musical experience. Utilizing heart rate variability data, the system dynamically adjusts the kind of songs recommended to better align the virtual performance space with the user's physiological state. Key components include accurate real-time HRV data capture, low-latency data transmission, and creating a responsive VR environment. Experiments demonstrated the system's effectiveness in enhancing user emotional state and immersion through personalized interactions. By harmonizing technology with human emotion, SingSense offers a novel platform that redefines interactive music experiences.

KEYWORDS

VR, Singing, Health

1. INTRODUCTION

Teen stress levels have reached unprecedented heights. According to the American Psychological Association in 2018, only 45% of Generation Z reported excellent or very good mental health, as opposed to 56% for millennials, 51% for Gen X, and 70% for boomers [1]. As reported by MedlinePlus, this teen stress commonly stems from school, relationships, and preparation for college [2]. Notably, according to the World Metrics Report 2024, only 8% of teens say their stress level is declining, underscoring the escalating need for effective stress alleviating mechanisms for adolescents[3].

However, more than half of these young adults do not receive the mental health services that they need, mostly due to the high cost, inaccessibility, and public stigma of these treatments [4][5][6]. These teenagers worry about being judged and fear that seeking therapy or taking medication might lead to social exclusion [5]. Therefore, there is a high social need for a cost-free, accessible, and entertaining stress alleviating mechanism, providing a fun yet effective method for adolescents to assess and alleviate stress.

David C. Wyld et al. (Eds): SEC, DMA, AIAPP, CYBI, CoSIT, SIGL, NLPML – 2025
pp. 187-208, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.150814

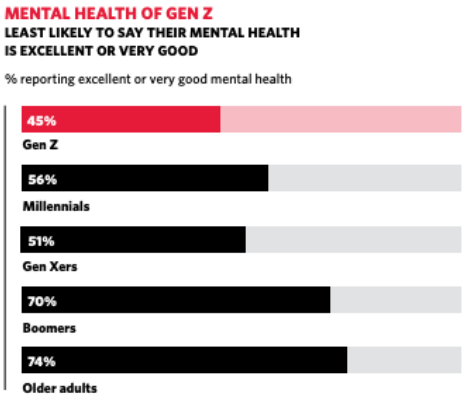

Figure 1. American Psychological Association, Stress In America Generation Z

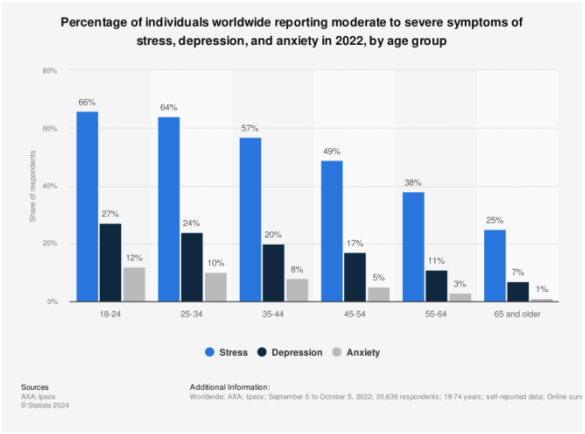

Figure 2. Worldwide; AXA; Ipsos; September 5 to October 5, 2022; 30,636 respondents; 18-74 years; self-reported data; Online survey

The long-term impact of this problem affects not only musicians but also individuals who use singing as a form of stress relief or therapy. SingSense has the potential to benefit music enthusiasts, VR users, and those seeking an emotionally engaging performance experience. With growing interest in both VR and biofeedback applications, a system like SingSense can redefine interactive music experiences for both casual and professional users.

The first methodology explored traditional VR singing applications that offer static environments without adaptive features. While providing a basic platform for singing, these applications lack personalization, leading to a less immersive experience.

The second approach examined VR systems incorporating basic user data, such as vocal pitch, to adjust certain aspects of the environment. Although an improvement over static systems, they often fail to integrate comprehensive biometric feedback, limiting the depth of interaction.

The third methodology, which SingSense employs, integrates real-time HRV data to create a fully adaptive VR environment. This approach offers a more immersive and personalized experience by aligning virtual elements with the user's physiological state, thereby enhancing emotional engagement and realism.

SingSense is a VR singing experience that integrates real-time biometric feedback to create an adaptive and immersive stress-relieving mechanism.

Singing has been shown to reduce stress effectively. A 2015 study found that singing in low-stress performance conditions led to a decrease in both cortisol and cortisone in the saliva of professional singers, indicating a reduction in stress.[7] Similarly, Virtual Reality(VR)-based interventions have demonstrated effectiveness in providing safe and controlled environments for stress management.[8] This experiment aims to test whether combining singing with a VR platform can aid in stress reduction.

Heart Rate Variability (HRV) are non-invasive biomarkers chosen in this experiment to quantify stress; in this experiment, the Root Mean Square of Successive Differences (RMSSD) is calculated to assess HRV [9]. RMSSD is the calculation of the root mean square of the sum of differences in the interbeat interval (IBI) (Fig. 3, Fig. 4). Essentially, it measures how different the interval between two heartbeats is from the interval before, and a higher RMSSD signifies lower stress levels as it signifies a better ability to respond to physiological stimuli [10].

$$\mathrm{RMSSD} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}(NN_{i+1} - NN_i)^2}$$

Figure 3. Formula of Root Mean Square of Successive Differences

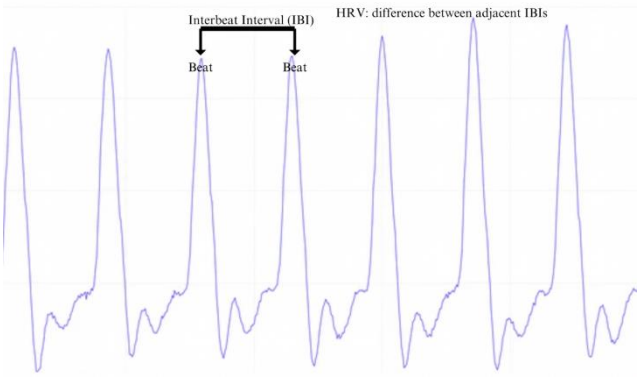

Figure 4. Visual definition of IBI and HRV

The results of this research could contribute to advancements in stress-alleviation strategies, particularly through musical and singing therapy integrated with VR technology. If successful, the platform could benefit students, athletes, and workers in managing stress and may also be extended to other groups, such as cancer patients or injured athletes, who experience high levels of mental stress.

This solution is unique compared to other stress alleviating mechanisms particularly due to its integration of VR technology and real time biometric data analysis. Unlike traditional methods,, SingSense immerses users in personalized VR environments that amplify relaxation while simultaneously monitoring their biological responses, making stress alleviation a lot more entertaining than it was. SingSense is also accessible online, allowing users to use it whenever and wherever they want to. Since SingSense provides the users scientific evidence that their

mental health has improved, the platform is able to create a positive feedback loop, where users are encouraged to engage regularly as they witness the direct benefits to their mental health.

The initial experiment put SingSense's HRV sensor in comparison with an ECG kit to determine the accuracy of the sensor. A test subject wore both devices, and HRV readings were recorded and compared across five trials. The sensor recorded a low error range of between 7.1% and 10%, confirming the reliability of the sensor. These results confirm the applicability of the sensor in tracking stress-related biometrics in performances and indicate its potential in offering useful data in further research and development for SingSense.

The second experiment measured users' HRV and emotional state before singing, while singing on SingSense, and after singing. The participants were equipped with a pulse sensor and a VR headset while singing. For 21 participants, the outcome showed HRV significantly reduced while singing and significantly increased afterward (overall from 44.3±7.5ms to 63.2±4.9ms), indicating recovery from physiological stress. Emotional state scores also increased following singing. Statistical significance was confirmed by a paired one-tailed t-test ($p<0.05$). These findings suggest that SingSense can reduce stress and improve mood, supported by biometric and self-reported evidence.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1.Designing a fully immersive VR environment

One of the main challenges in creating SingSense is designing a fully immersive VR environment that effectively simulates a jazz club atmosphere. The primary issues involve graphical rendering, user interaction, and ensuring smooth performance across different VR headsets. Performance optimization is critical, as VR applications must maintain a high frame rate to prevent motion sickness and provide a seamless experience.
To address these issues, a game engine like Unity could be used to build a detailed but optimized environment. Efficient rendering techniques, such as level-of-detail (LOD) adjustments and texture streaming, could be implemented to maintain high-quality visuals without sacrificing performance. Additionally, intuitive user interaction systems could be designed, allowing singers to navigate menus, select songs, and interact with their surroundings naturally using VR controllers or hand-tracking technology.

2.2. Integrating real-time singing data

Integrating real-time singing data presents another significant challenge. The system needs to process vocal input, determine pitch accuracy, and synchronize it with background music while ensuring minimal latency. One issue is the variability in microphone quality across different VR headsets, which can impact sound clarity and pitch detection accuracy.

To tackle this, machine learning-based pitch detection algorithms could be employed to analyze vocal input with high precision. A combination of FFT (Fast Fourier Transform) and deep learning models could improve accuracy while filtering out background noise. Additionally, cloud-based or on-device processing techniques could be explored to ensure minimal lag while analyzing and visualizing the singer’s performance in real-time.

2.3. Heart rate variability

Figure 5. Sensor

Heart rate variability (HRV) is a crucial component of SingSense, but accurately measuring and interpreting HRV data in real-time is complex. HRV is influenced by numerous factors, including movement, stress, and external distractions, which can introduce inconsistencies in readings. Additionally, integrating HRV data into the VR experience requires a reliable connection between the heart rate sensor and the VR system.

One approach to solving this challenge is implementing advanced signal processing techniques to filter out noise and artifacts in HRV readings. Setting up a dynamic threshold system could help refine the accuracy of HRV interpretation. Additionally, wifi communication with a unified database system could be optimized to ensure seamless real-time data transfer between the HR sensor and the VR application. By refining these techniques, SingSense could provide a smooth, responsive, and truly immersive experience that reacts dynamically to the singer's physiological state.

3. SOLUTION

SingSense integrates three major components to create an interactive VR singing experience: the VR application, Firebase for data synchronization, and HRV/BPM pulse sensors. The system starts with the user launching the VR app, which provides an immersive jazz club setting where they can select a song. The pulse sensor records real-time biometric data, which is transmitted to Firebase. Firebase acts as a middleware, ensuring seamless data flow between the sensor and the Unity-based VR game. The VR environment dynamically adjusts elements such as lighting and tempo based on HRV and BPM data, creating a fully immersive and responsive experience.

The physical product includes any VR headset and a Pulse sensor attached to the headset's controller. As the user constantly holds the controller, real time heart rate and HRV will be measured, recorded, and sent to the virtual component using an Arduino board.

The virtual component of the design is created using platforms such as Unity Hub, Visual Studio Code, and Arduino IDE. In a relaxing virtual Jazz Club, the user faces UI Panels. A login/signup panel will be included for creating accounts and recording data from the user used to assess the overall effectiveness of the application. The user then scrolls sliders about their happiness, energy level, preferred song tempo, preferred song intensity, and preferred song pitch range. These values are needed for the system to recommend the best song for the user to sing at the moment in order to best reduce their stress level. The recommendation system also considers the user's

current HRV compared to their baseline HRV, estimating their stress level and making the best song recommendation. The song is then shown to the user, and the user can then choose to retry or sing the song. Another panel will allow the user to pick the number of audiences(dummies) they want to maximize their comfort, and will be implemented in the scene. As the user clicks start singing, 3 other panels respectively showing the lyrics, real-time biological parameters, and song information will be presented. Lastly, when the user finishes singing, another panel will showcase their average heart rate before, during, and after singing.

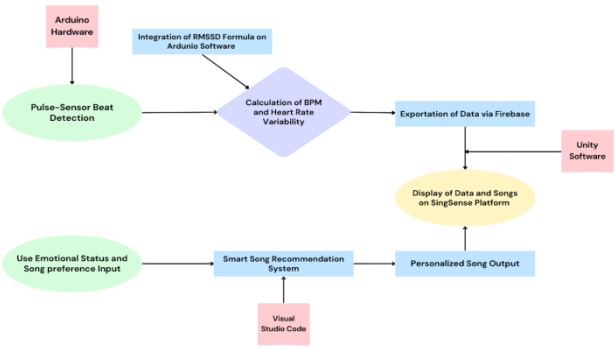

Figure 6. Internal Mechanism of the SingSense Platform

Figure 7. Flowchart of UI Boards Shown to the User

The VR app, developed in Unity, creates an interactive jazz club environment where users perform. It features real-time rendering, audio synchronization, and environmental adjustments. The app relies on Unity's physics engine and graphical rendering capabilities to deliver a realistic experience.

Figure 8. Screenshot of the lyrics

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class PitchDetector : MonoBehaviour
{
    public MicrophoneInput microphoneInput;
    // higher value increases frequency resolution but requires more
processing power
    private int sampleSize = 1024;
    // an array to store raw audio samples from the microphone
    private float[] audioSamples;
    // an array to store the frequency spectrum
    private float[] spectrum;
    // fundamental frequency
    private float fundamentalFrequency;

    void Start()
    {
        audioSamples = new float[sampleSize];
        spectrum = new float[sampleSize];
    }

    void Update()
    {
        AnalyzeAudio();
    }
```

```
    void AnalyzeAudio()
    {
        // // get audio from the microphone
        // microphoneInput.audioSource.GetOutputData(audioSamples, 0);

        // // calculate the FFT (Fast Fourier Transform) to get the
frequency data
        // FFT(audioSamples, spectrum);

        microphoneInput.audioSource.GetSpectrumData(spectrum, 0,
FFTWindow.BlackmanHarris);

        // perform autocorrelation to find the fundamental frequency
        float sum = 0;
        float[] autocorrelation = new float[sampleSize];

        // i is the lag
        for (int i = 0; i < sampleSize; i++) {
            sum = 0;

            // for each lag 'i', calculate the autocorrelation
            for (int j = 0; j < sampleSize - i; j++) {
                sum += audioSamples[j] * audioSamples[j+i];
            }

            autocorrelation[i] = sum;
        }
```

```
        // find the first peak in the autocorrelation
        // int peakIndex = 1;
        float peakValue = autocorrelation[1];

        for (int i = 2; i < sampleSize; i++) {
            if (autocorrelation[i] > peakValue) {
                peakValue = autocorrelation[i];
                // peakIndex = i;
            }
        }

        // calculate the fundamental frequency
        fundamentalFrequency = 44100f / peakValue;
        Debug.Log($"Fundamental Frequency: {fundamentalFrequency} Hz");

    }
}
```

Figure 9. Screenshot of code 1

The `PitchDetector` class in Unity is designed to analyze audio input from a microphone to determine the fundamental frequency of the sound, which is essential for applications like musical note recognition. Upon starting the application, the `Start` method initializes two arrays, `audioSamples` and `spectrum`, each with a size defined by `sampleSize` (1024). These arrays store raw audio data and frequency spectrum data, respectively.

In the `Update` method, which runs once per frame, the `AnalyzeAudio` method is invoked to process the incoming audio. Within `AnalyzeAudio`, the application captures the frequency spectrum data from the microphone's audio source using Unity's `GetSpectrumData` method with a Blackman-Harris window, which helps in reducing spectral leakage.

The method then performs autocorrelation to identify the fundamental frequency. Autocorrelation involves multiplying the audio signal by a time-shifted version of itself and summing the result over a range of lags. This process helps in detecting repeating patterns in the signal, which correspond to the fundamental frequency.

After computing the autocorrelation for various lags, the code searches for the first significant peak in the autocorrelation array, which indicates the fundamental period of the input signal. The fundamental frequency is then calculated by dividing the sampling rate (44100 Hz) by the position of this peak. Finally, the detected fundamental frequency is output to the console for debugging purposes.

This approach allows for real-time pitch detection, which is crucial for applications like musical note recognition or tuning systems. However, it's important to note that while autocorrelation is effective for pitch detection, it can be computationally intensive and may require optimization for real-time performance. Additionally, factors such as noise and harmonic distortion can affect the accuracy of the detected pitch.

Firebase is used to consolidate biometric data, ensuring real-time updates between the HRV sensor and Unity. Firebase's real-time database provides a low-latency solution for data transfer, ensuring a seamless connection between physiological responses and in-game adaptations.

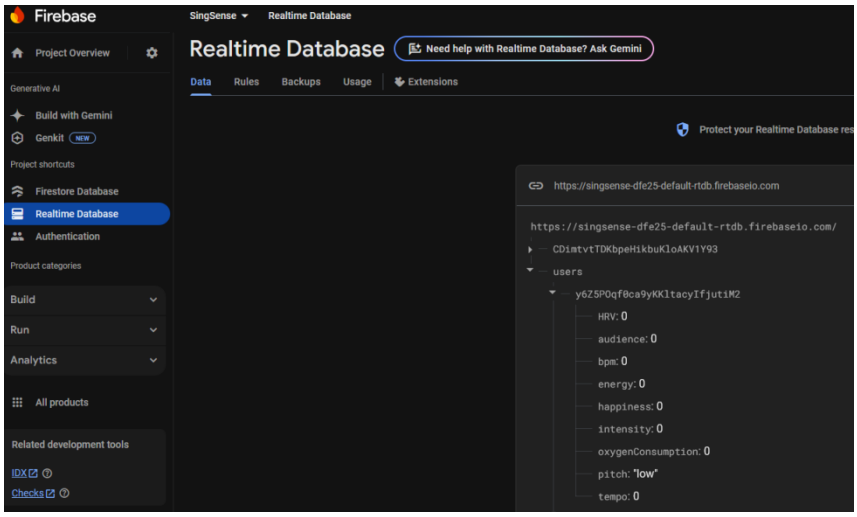

Figure 10. Screenshot of firebase

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Firebase;
using Firebase.Database;
using Firebase.Auth;



namespace Managers
{
    public class AuthManager: MonoBehaviour
    {
        public FirebaseAuth auth;
        public FirebaseUser user;
        public DatabaseReference dbReference;
        public static bool loggedIn;


        private void Awake()
        {
            InitializeFirebase();
        }


        private void InitializeFirebase()
        {
            auth = FirebaseAuth.DefaultInstance;
            dbReference = FirebaseDatabase.DefaultInstance.RootReference;
            auth.StateChanged += AuthStateChanged;
            AuthStateChanged(this, null);
        }


        private void AuthStateChanged(object sender, System.EventArgs
eventArgs)
        {
            if (auth.CurrentUser != user)
            {
                loggedIn = user != auth.CurrentUser && auth.CurrentUser !=
null;
                if (!loggedIn && user != null)
                {
```

```
                    Debug.Log("Signed out " + user.UserId);
                }
                user = auth.CurrentUser;
                if (loggedIn)
                {
                    Debug.Log("Signed in " + user.UserId);
                }
            }
        }


        public void Login(string email, string password)
        {
            auth.SignInWithEmailAndPasswordAsync(email,
password).ContinueWith(task =>
            {
                if (task.IsCanceled)
                {
                    Debug.LogError("SignInWithEmailAndPasswordAsync was
canceled.");
                    return;
                }
                if (task.IsFaulted)
                {
                    Debug.LogError("SignInWithEmailAndPasswordAsync
encountered an error: " + task.Exception);
                    return;
                }
                Firebase.Auth.AuthResult result = task.Result;
```

```
                Debug.LogFormat("User signed in successfully: {0} ({1})",
                    result.User.DisplayName, result.User.UserId);
            });
        }


        public void SignUp(string email, string password)
        {
            auth.CreateUserWithEmailAndPasswordAsync(email,
password).ContinueWith(task =>
            {
                if (task.IsCanceled)
                {
                    Debug.LogError("CreateUserWithEmailAndPasswordAsync was
canceled.");
                    return;
                }
                if (task.IsFaulted)
                {
                    Debug.LogError("CreateUserWithEmailAndPasswordAsync
encountered an error: " + task.Exception);
                    return;
                }
                // Firebase user has been created.
                Firebase.Auth.AuthResult result = task.Result;
                Debug.LogFormat("Firebase user created successfully: {0}
({1})",
                    result.User.DisplayName, result.User.UserId);
                user = result.User;
                CreateUserOnRTDB(user.UserId, email);
```

```
        });
    }

    public void SignOut()
    {
        auth.SignOut();
        auth.StateChanged -= AuthStateChanged;
        loggedIn = false;
    }

    void CreateUserOnRTDB(string userId, string email)
    {
        Dictionary<string, object> user = new Dictionary<string,
object>();
        user["audience"] = 0;
        user["bpm"] = 0;
        user["breathRate"] = 0;
        user["energy"] = 0;
        user["happiness"] = 0;
        user["intensity"] = 0;
        user["oxygenConsumption"] = 0;
        user["pitch"] = "low";
        user["tempo"] = 0;
        dbReference.Child("users").Child(userId).SetValueAsync(user);
        Debug.Log("User created on RTDB");
    }
}
```

Figure 11. Screenshot of code 2

The `AuthManager` class in Unity manages user authentication and real-time database interactions using Firebase. Upon the `Awake` event, the `InitializeFirebase` method is invoked to set up the Firebase Authentication (`auth`) and Database (`dbReference`) instances. It also subscribes to the `auth.StateChanged` event to monitor authentication state changes.

The `AuthStateChanged` method checks if the current user (`auth.CurrentUser`) differs from the previously stored user (`user`). If there's a change, it updates the `loggedIn` status and logs the sign-in or sign-out event accordingly.

The `Login` method allows users to sign in with an email and password by calling `SignInWithEmailAndPasswordAsync`. It handles potential errors and logs the result upon successful authentication.

Similarly, the `SignUp` method enables new user registration using `CreateUserWithEmailAndPasswordAsync`. Upon successful account creation, it calls `CreateUserOnRTDB` to initialize the user's data in the Firebase Realtime Database.
The `SignOut` method signs the user out and unsubscribes from the `AuthStateChanged` event, updating the `loggedIn` status to `false`.

The `CreateUserOnRTDB` method initializes a new user's data in the Realtime Database with default values for various metrics, such as `audience`, `bpm`, and `energy`. It uses a dictionary to structure the data and sets it asynchronously in the database under the user's unique ID.

This class ensures seamless authentication and user data management within the Unity application, leveraging Firebase's services for backend support.

The HRV/BPM sensors capture heart rate variability and beats per minute, sending data to Firebase. Signal processing techniques filter noise and inconsistencies, ensuring accurate biometric tracking and synchronization with VR elements.

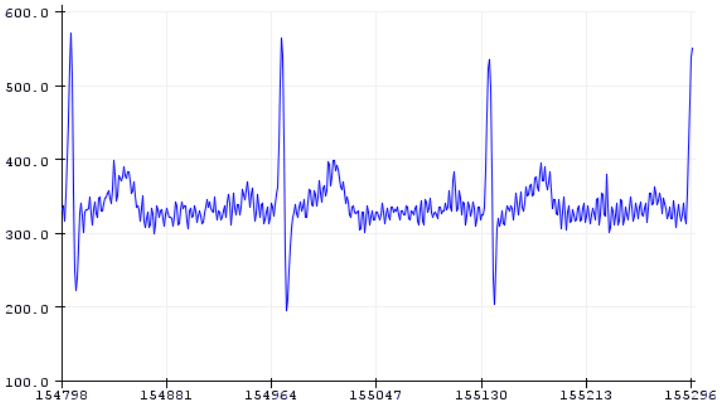

Figure 12. Screenshot of game 3

```
#include <PulseSensorPlayground.h>  // Includes the PulseSensorPlayground
Library.

//  Variables

const int PulseWire = 0;       // PulseSensor PURPLE WIRE connected to
ANALOG PIN 0

const int LED = LED_BUILTIN;          // The on-board Arduino LED, close to
PIN 13.

int Threshold = 550;           // Determine which Signal to "count as a
beat" and which to ignore.

                               // Use the "Gettting Started Project" to
fine-tune Threshold Value beyond default setting.

                               // Otherwise leave the default "550" value.

const int SAMPLE_INTERVAL = 20; //20ms sample interval
```

```
PulseSensorPlayground pulseSensor;  // Creates an instance of the
PulseSensorPlayground object called "pulseSensor"


// HRV variables

const int MAX_RR_INTERVALS = 300; // Max number of intervals to store

unsigned long rrIntervals[MAX_RR_INTERVALS]; // Array to store RR intervals

int rrIndex = 0; // Index of the last RR interval

float rmssd = -1; // Root mean square of successive differences

const int RMSSD_WINDOW_SIZE = 50; // Window size for RMSSD calculation


// Beat detection variables

unsigned long lastBeatTime = 0;

int IBI = 0;

int lastIBI = 0;        //inter beat interval

int P = 0;          //peak value of IBI waveform

int T = 10000;        // trough value of IBI waveform

bool goingUp = true;

float highBPM = 0;

float lowBPM = 0;

float BPM = 0;

float HRV = 0;

long runningTotal = 1;

float mean = 0;

int amp = 0;

bool firstBeat = true;
```

```
// Moving Average Filter Variables

const int MA_WINDOW_SIZE = 10; // Window size for moving average filter

int maWindow[MA_WINDOW_SIZE] = {0}; // Array to store window values

int maIndex = 0; // Index of the last value in the window

int maTotal = 0; // Total of the values in the window

float maValue = 0; // Moving average value



// Dynamic Threshold Variables

float dynamicThreshold = 550.0;           // Determine which Signal to
"count as a beat" and which to ignore.

const float MAX_THRESHOLD = 800.0;        // Maximum threshold value

const float MIN_THRESHOLD = 400.0;        // Minimum threshold value



// IBI Validity Variables

const int MIN_IBI = 30; // Minimum IBI value (300/5 given sample rate of 20
is about 190 BPM)

const int MAX_IBI = 200; // Maximum IBI value (300/5 given sample rate of 20
is about 30 BPM)

const float IBI_OUTLIER_THRESHOLD = 0.20; // IBI outlier threshold



unsigned long median(unsigned long arr[], int n) {

  // make a copy of the array

  unsigned long sorted[n];


  memcpy(sorted, arr, n * sizeof(unsigned long));


  // insertion sort

  for (int i =1; i < n; i++) {

    unsigned long key = sorted[i];

    int j = i - 1;

    while (j >= 0 && sorted[j] > key) {

      sorted[j + 1] = sorted[j];

      j--;

    }

    sorted[j + 1] = key;

  }


  // return the median

  if (n % 2 == 1) {

    return sorted[n / 2];

  } else {

    return (sorted[n/2 - 1] + sorted[n/2]) / 2;

  }

}


bool isIBIoutlier(unsigned long currentIBI, unsigned long refIBI, float
threshold) {

  // unsigned long diff = abs(currentIBI - refIBI);

  unsigned long diff = (currentIBI > refIBI) ? currentIBI - refIBI : refIBI
- currentIBI;

  Serial.print(diff);

  Serial.print(" > ");

  Serial.println(threshold * refIBI);

  Serial.print("Threshold: ");

  Serial.print(threshold);

  Serial.print(" * ");

  Serial.print("Reference IBI: ");

  Serial.println(refIBI);

  return (diff > threshold * refIBI);

}


// calculate RMSSD function

void calculateRMSSD() {

  int count = min(rrIndex, RMSSD_WINDOW_SIZE);


  if (count < 2) {

    rmssd = -1; // Not enough data to calculate RMSSD

    return;

  }


  float sumSqDiff = 0;

  int validDifferences = 0;


  // temp array to hold the last 'count' IBIs:

  unsigned long tempIBIs[count];

  int startIndex = (rrIndex < RMSSD_WINDOW_SIZE) ? 0 : rrIndex -
RMSSD_WINDOW_SIZE;

  for (int i = 0; i < count; i++) {

    tempIBIs[i] = rrIntervals[startIndex + i];

  }


  for (int i = 0; i < count - 1; i++) {

    long diff = (tempIBIs[i+1] > tempIBIs[i]) ? tempIBIs[i + 1] -
tempIBIs[i] : tempIBIs[i] - tempIBIs[i + 1];
```

```
    Serial.print("Difference: ");
    Serial.println(diff);

    if (diff < MIN_IBI || diff > MAX_IBI) {
      continue;
    }

    sumSqDiff += diff * diff;
    validDifferences++;
  }

  if (validDifferences > 0) {
    rmssd = sqrt(sumSqDiff / validDifferences);
  } else {
    rmssd = -1;
  }
}

void setup() {

  Serial.begin(9600);          // For Serial Monitor

  // Configure the PulseSensor object, by assigning our variables to it.
  pulseSensor.analogInput(PulseWire);
  pulseSensor.blinkOnPulse(LED);       //auto-magically blink Arduino's LED
with heartbeat.
  pulseSensor.setThreshold(Threshold);

  // Initalize Moving Average Buffer
  for (int i = 0; i < MA_WINDOW_SIZE; i++) {
    int initialRead = analogRead(PulseWire);
    maWindow[i] = initialRead;
    maTotal += initialRead;
    delay(10);
  }
  maValue = maTotal / (float)MA_WINDOW_SIZE;
  maIndex = 0;

  // Double-check the "pulseSensor" object was created and "began" seeing a
signal.
   if (pulseSensor.begin()) {
    Serial.println("We created a pulseSensor Object !");  //This prints one
time at Arduino power-up,  or on Arduino reset.
  }
}

void loop() {
 // to do: add RMSSD
 int rawSignal = analogRead(PulseWire);  // Read the PulseSensor's value.
 if (rawSignal > Threshold) {
  digitalWrite(LED, HIGH);
 } else {
  digitalWrite(LED, LOW);
  }
```

```
// write to plotter
// Serial.println(rawSignal);


// Moving Average Filter
maTotal -= maWindow[maIndex];
maWindow[maIndex] = rawSignal;
maTotal += maWindow[maIndex];
maIndex = (maIndex + 1) % MA_WINDOW_SIZE;
maValue = maTotal / (float)MA_WINDOW_SIZE;


if (pulseSensor.sawStartOfBeat()) {            // Constantly test to see if
"a beat happened".
unsigned long currentTime = millis();
BPM = pulseSensor.getBeatsPerMinute();  // Calls function on our
pulseSensor object that returns BPM as an "int".
                                               // "myBPM" hold this BPM
value now.


// HRV Calculation
if (!firstBeat) {
  IBI = currentTime - lastBeatTime;


  if (IBI >= MIN_IBI && IBI <= MAX_IBI) {
    bool reject = false;


    // Check for IBI outliers
    if (rrIndex >= 10) {
      unsigned long prevIBIs[10];


      for (int i = 0; i < 10; i++) {
        prevIBIs[i] = rrIntervals[rrIndex - 10 + i];
      }


      unsigned long medianIBI = median(prevIBIs, 10);
      Serial.print("Median IBI of the last 10: ");
      Serial.println(medianIBI);
      if (isIBIoutlier(IBI, medianIBI, IBI_OUTLIER_THRESHOLD)) {
        reject = true;
        Serial.print("Rejected IBI due to outlier: ");
        Serial.println(IBI);
      }
    }


    if (!reject) {
      // Store the RR interval in the buffer
      Serial.print("Storing IBI: ");
      Serial.println(IBI);
      if (rrIndex < MAX_RR_INTERVALS) {
        rrIntervals[rrIndex] = IBI;
        rrIndex++;
      } else {
        // Shift the buffer to the left
        for (int i = 1; i < MAX_RR_INTERVALS; i++) {
          rrIntervals[i-1] = rrIntervals[i];
```

```
        }
        rrIntervals[MAX_RR_INTERVALS-1] = IBI;
      }
      // printArray(rrIntervals, rrIndex);
    }

    // Calculate RMSSD
    calculateRMSSD();

   } else {
    // Serial.println("Invalid IBI: " + String(IBI));
   }
  } else {
   firstBeat = false;
  }
  lastBeatTime = currentTime;

 Serial.print("BPM: ");                        // Print phrase "BPM: "
 Serial.println(BPM);
 Serial.print("RMSSD: ");                      // Print phrase "RMSSD: "
 if (rmssd >= 0) {
  Serial.println(rmssd);                     // Print the value inside of
RMSSD.
 } else {
  Serial.println("Not enough data");
 }
}

 delay(SAMPLE_INTERVAL);                     // considered best practice in
a simple sketch.


}
```

Figure 13. Screenshot of code 3

The provided Arduino code utilizes the PulseSensor Playground library to monitor heart rate and compute Heart Rate Variability (HRV) metrics, specifically the Root Mean Square of Successive Differences (RMSSD). The `PulseSensorPlayground` object is configured to read analog input from a pulse sensor connected to pin A0. In the `setup()` function, the sensor is initialized, and a moving average filter is prepared to smooth incoming data.

Within the `loop()` function, the code reads the raw signal from the pulse sensor and applies a moving average filter to reduce noise. When a heartbeat is detected (`pulseSensor.sawStartOfBeat()` returns `true`), the current time is recorded, and the time interval between beats (Inter-Beat Interval or IBI) is calculated. The code checks for valid IBI ranges and filters out outliers by comparing the current IBI to the median of the previous ten IBIs. Valid IBIs are stored in an array for HRV analysis.

The `calculateRMSSD()` function computes the RMSSD by taking the square root of the mean of the squared differences between successive IBIs, providing an index of HRV. This metric is valuable for assessing autonomic nervous system activity and overall cardiovascular health.

Throughout the process, the code outputs the current BPM and RMSSD values to the serial monitor, allowing for real-time monitoring and analysis of heart rate and its variability.

This implementation demonstrates how Arduino can be used in conjunction with the PulseSensor Playground library to perform real-time heart rate monitoring and HRV analysis, which can be applied in various health and fitness applications.

4. EXPERIMENT

4.1. Experiment 1

A potential blind spot in SingSense is the accuracy and responsiveness of real-time heart rate variability (HRV) data integration during user performances. Ensuring that the sensors remain accurate is crucial for further experiments.

To assess this, an ECG kit would be worn by the subject alongside the HRV sensor. The system would record the biometric data from both devices, and the data would be compared and analyzed to ensure accurate biometric readings.

Participant ID	ECG kit HRV (ms)	Sensor HRV (ms)	% Error
1	50	45	10
2	60	55	8.3
3	55	50	9.1
4	65	60	7.7
5	70	65	7.1

Figure 14. Artistic Autists Navigation Survey Data

The HRV calculated by the sensor was relatively accurate compared to the HRV measured by an ECG kit, as the % error ranges between 7-10%. This showcases the accuracy of the HRV testing by the sensor used by singsense. While there are still certain errors measured, the sensor deployed by SingSense provided accurate and precise readings, strengthening the immersive experience and preparing the platform for further experimentation.

4.2. Experiment 2

This experiment measures the changes in users' HRV before, during, and after singing on SingSense.

To start the experiment, the subject put on the pulse sensor, and HRV was measured and calculated for 15 seconds. The average value of HRV and the user's self-reported emotional state were then recorded.

Then, the user put on the VR headset and used the program. Their fingertips continuously touch the pulse sensor, which is wrapped around the VR controller. The HRV values during singing were averaged and recorded.

3 minutes after the song ended, in order to control potential errors since physical activities such as singing affects HRV temporarily, the HRV and self-reported emotional state were recorded again.

The above procedure was seen as one trial, and three trials were conducted for each subject, with 21 subjects tested in total. Subjects informed consent; subjects varied in age, race, and gender.

When organizing data, a paired, 1-tailed t-test was used between every run to calculate statistical significance between runs. Average Absolute Deviation (AAD) was calculated and applied in error bars.

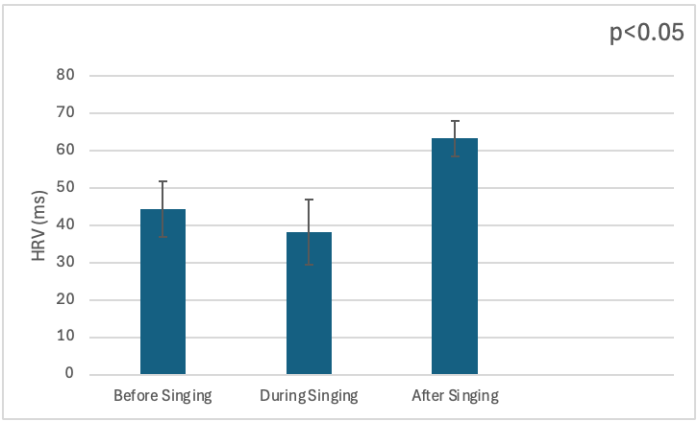

Figure 15. Average HRV of baseline, before, during, and after subject sings on SingSense. n=21, p<0.05, and error bars represent AAD.

The HRV of subjects before singing (44.3±7.5ms) is significantly higher than HRV while singing (38.2±8.7ms); 3 minutes after singing, the HRV of subjects rises significantly, being the highest of all four runs (63.2±4.9ms). Once again, the AAD of the HRV of subjects during the usage of SingSense is the highest of all four runs.

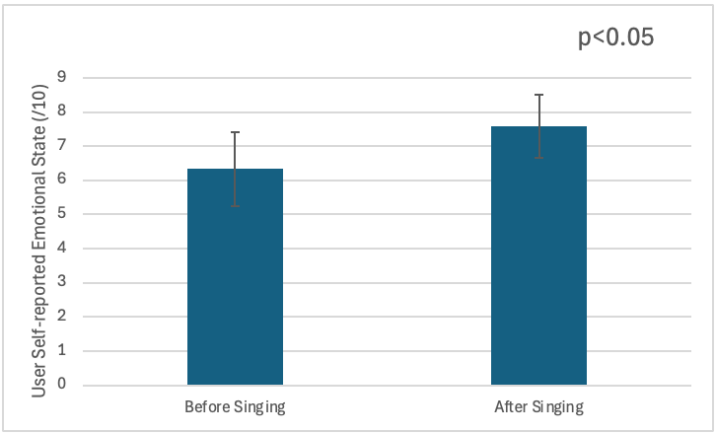

Figure 16. Average user self-reported emotional state (/10) before and after the subject sings on SingSense. n=21, p<0.05, and error bars represent AAD.

Users' average self-reported emotional state on a scale from 0 to 10 after using SingSense is significantly higher than before using the platform (7.6±0.9 and 6.3±1.1).

The decrease in HRV while singing can be explained by the shift from the parasympathetic nervous system (dominant when resting) to the sympathetic nervous system (dominant during activity), leading to more regular, fast-paced heartbeats, decreasing HRV while increasing BPM [14]. Although singing often is not considered a regular form of exercise, it requires high activity of pulmonary receptors, lung mechanical effects, and adrenal response, leading to the shift in the nervous system [14].

The significant increase in HRV 3 minutes after singing is the post-singing recovery, where the increase demonstrates an increase in cardiovascular ability to respond to external stimuli and a decrease in stress level [15].
Figure 14 provides additional insight into the users' self-reported emotions. The significant increase in emotional state illustrates a subjective decrease in stress level and an increase in mental well-being after singing, along with the objective evidence shown by HRV.

5. RELATED WORK

A study by Aufegger and Wasley (2020) explored the impact of virtual reality (VR) feedback on musicians' performance confidence, perceived performance quality, state anxiety, and heart rate variability (HRV). Musicians were exposed to both positive and negative simulated feedback within a VR environment to assess its influence on their psychological and physiological responses. The study found that VR feedback could significantly affect musicians' mental attitudes and physical responses during performances. However, the research primarily focused on the effects of external feedback rather than integrating real-time biometric data to modify the VR environment. SingSense builds upon this by using live HRV data to dynamically adjust the virtual setting, creating a more personalized and responsive experience.

Erickson (2021) examined the use of mobile applications and biofeedback in voice pedagogy, utilizing visual biofeedback to aid singers in navigating challenging vocal regions. The study demonstrated that visual biofeedback could enhance vocal performance by providing singers with real-time data on their vocal output. While effective in improving technical aspects of singing, this approach does not incorporate VR or create an immersive performance environment. SingSense advances this concept by embedding biofeedback within a VR setting, allowing singers to experience a simulated performance space that reacts to their physiological states, thereby enhancing both technical skills and emotional engagement.

A review by Wiśniewska and Sawicki (2021) analyzed the effects of music on heart rate variability, focusing on passive music therapy and its impact on the autonomic nervous system. The review concluded that music could influence HRV, suggesting potential therapeutic applications. However, the studies reviewed primarily involved passive listening experiences and did not explore active engagement or the use of VR environments. SingSense addresses this gap by combining active musical participation with real-time HRV monitoring within a VR context, offering an innovative approach that merges performance, biofeedback, and immersive technology for a holistic user experience.

6. CONCLUSIONS

While SingSense offers an innovative integration of biometric feedback into a VR singing environment, several limitations exist. Accurately capturing and interpreting heart rate variability (HRV) in real-time can be challenging due to potential signal noise and artifacts caused by user movement. Implementing advanced signal processing techniques and machine learning algorithms could enhance data accuracy. Additionally, the system's responsiveness may be affected by latency in data transmission between the biometric sensors and the VR application. Optimizing data communication protocols and exploring edge computing solutions might mitigate this issue. The current VR environment, while immersive, may not cater to all user preferences or accessibility needs. Expanding the range of virtual settings and incorporating customizable features could broaden user appeal. Furthermore, the reliance on specific hardware for biometric sensing may limit accessibility for some users. Developing compatibility with a

wider range of devices or utilizing built-in sensors in existing VR hardware could enhance inclusivity. Given more time, these improvements would be prioritized to enhance user experience and system performance.

SingSense represents a pioneering step in merging biometric data with virtual reality to create a dynamic and personalized singing experience. By addressing current limitations and implementing proposed improvements, SingSense has the potential to redefine interactive music applications, offering users a deeply immersive and responsive platform that harmonizes technology with human emotion.

REFERENCES

[1] Pope, Kenneth S., and Valerie A. Vetter. "Ethical dilemmas encountered by members of the American Psychological Association: A national survey." American Psychologist 47.3 (1992): 397.

[2] Madaan, Aastha, Wanming Chu, and Subhash Bhalla. "Vishue: Web page segmentation for an improved query interface for medlineplus medical encyclopedia." Databases in Networked Information Systems: 7th International Workshop, DNIS 2011, Aizu-Wakamatsu, Japan, December 12-14, 2011. Proceedings 7. Springer Berlin Heidelberg, 2011.

[3] Henry, James P. "Biological basis of the stress response: Address upon accepting the Hans Selye Award from the American institute of stress in Montreux, Switzerland, February 1991." Integrative physiological and behavioral science 27.1 (1992): 66-83.

[4] Andrade, Laura Helena, et al. "Barriers to mental health treatment: results from the WHO World Mental Health surveys." Psychological medicine 44.6 (2014): 1303-1317.

[5] Tulk, Christine, et al. "The impact of COVID-19 on the mental health and substance use health (MHSUH) workforce in Canada: a mixed methods study." Human Resources for Health 21.1 (2023): 9.

[6] Patel, Vikram, et al. "Mental health of young people: a global public-health challenge." The lancet 369.9569 (2007): 1302-1313.

[7] Fancourt, Daisy, Lisa Aufegger, and Aaron Williamon. "Low-stress and high-stress singing have contrasting effects on glucocorticoid response." Frontiers in psychology 6 (2015): 1242.

[8] Meshkat, Shakila, et al. "Virtual reality and stress management: A systematic review." Cureus 16.7 (2024).

[9] van Ravenswaaij-Arts, Conny MA, et al. "Heart rate variability." Annals of internal medicine 118.6 (1993): 436-447.

[10] Kim, Hye-Geum, et al. "Stress and heart rate variability: a meta-analysis and review of the literature." Psychiatry investigation 15.3 (2018): 235.

[11] Somayaji, Kamila, et al. "Acute effects of singing on cardiovascular biomarkers." Frontiers in Cardiovascular Medicine 9 (2022): 869104.

[12] Kang, Jing, Austin Scholp, and Jack J. Jiang. "A review of the physiological effects and mechanisms of singing." Journal of Voice 32.4 (2018): 390-395.

[13] North, Max M., Sarah M. North, and Joseph R. Coble. Virtual reality therapy: An innovative paradigm. Ipi Press, 1996.

[14] Rothbaum, Barbara Olasov, et al. "A controlled study of virtual reality exposure therapy for the fear of flying." Journal of consulting and Clinical Psychology 68.6 (2000): 1020.

[15] ADJORLU, Ali, and Stefania SERAFIN. "SINGING FOR ANXIETY: EXPLORING THE POTENTIALS OF SINGING IN IMMERSIVE VIRTUAL REALITY TO HELP CHILDREN WITH SOCIAL ANXIETY." Perspectives on immersion through laser doppler vibrometry (2024): 158.

[16] Agarwal, Ayush, et al. "Biometrics in extended reality: a review." Discover Artificial Intelligence 4.1 (2024): 81.

[17] Tosti, Beatrice, et al. "Neurofeedback Training Protocols in Sports: A Systematic Review of Recent Advances in Performance, Anxiety, and Emotional Regulation." Brain Sciences 14.10 (2024): 1036.

[18] Aufegger, Lisa, and David Wasley. "Virtual reality feedback influences musicians' physical responses and mental attitude towards performing." Music Med 12 (2020): 157-166.

[19] Erickson, Heidi Moss. "Mobile Apps and Biofeedback in Voice Pedagogy." Journal of Singing 77.4 (2021).

[20] [20] Parizek, D., et al. "The effect of music on heart rate variability." Acta Medica Martiniana 21.1 (2021): 1-8.