公立はこだて未来大学 2024 年度 システム情報科学実習
グループ報告書

Future University Hakodate 2024 Systems Information Science Practice
Group Report

プロジェクト名

DLITE3：境界なく人々の生活を支援する技術

Project Name

DLITE3：Technology that supports people's lives without boundaries

グループ名

B-聴覚支援グループ

Group Name

B-Hearing Support Group

プロジェクト番号 / Project No.

22

プロジェクトリーダ / Project Leader

金子康一　Kaneko Koichi

グループリーダ / Group Leader

佐々木大聖　Sasaki Taisei

グループメンバ / Group Member

北清良菜　Kitase Rana
櫻田空大　Sakurada Sorata
佐々木大聖　Sasaki Taisei

指導教員

三上貞芳　伊藤精英　島影圭佑　宮本エジソン正

Advisor

Mikami Sadayoshi　Ito Kiyohide　Shimakage Keisuke　Miyamoto, Edson T.

提出日 / Date of Submission

2024 年 1 月 21 日　January 21, 2025

概要

本プロジェクトでは、「視覚や聴覚に頼れない状況で役立つ装置の開発」をコンセプトとし、障害者が抱える問題を当事者目線で検討し、実用的な装置の開発に取り組んできた。頼れない感覚を別の手段で補うことで、不便を解消し、安全で快適な生活を支援することを目指している。聴覚障害や視覚障害、色覚の障害者を対象とした４つのグループに分かれ、それぞれ、特定の言葉に反応するデバイス、画像の色をユニバーサルデザインに変換するアプリ、自力で避難することが難しい人のための補助デバイス、障害者が自然を楽しむためのデバイスの開発を行っている。本グループは聴覚障害支援班として、聴覚に障害を持つ方々の私生活を支援するためのデバイスを開発している。

キーワード　障害者支援, 聴覚補助, 音声認識, Raspberry Pi, 音響モデル, 言語モデル, 振動通知, ユニバーサルデザイン

（※文責：佐々木大聖）

Abstract

Under the concept of "developing devices that are useful in situations where one cannot rely on sight or hearing," this project examines the problems faced by people with disabilities from the perspective of the people concerned, to develop practical devices. By supplementing unreliable senses with other means, the project aims to eliminate inconvenience and support safe and comfortable living. The project is divided into four team targeting people with hearing disabilities, visual disabilities, and color blindness. Each team is developing devices that respond to specific words and sounds, applications that convert the color of images to universal design, assistive devices for people who have difficulty evacuating on their own, and devices that allow people with disabilities to enjoy nature. As the Hearing Impairment Support Group, this group develops devices to assist people with hearing impairment in their private lives.

（※文責：佐々木大聖）

目次

第1章　はじめに

1.1　背景

現代社会において、音声を介した情報伝達は主要な手段とされている。そのため、聴覚障害者にとって情報へのアクセスやコミュニケーションに多くの困難が生じている。この問題は、生まれつきの聴覚障害者だけでなく、高齢者や中途失聴者にも該当する。本研究においては、「日常生活において聞き取ることが難しいと感じる人々」を聴覚障害者として定義し、対象者の範囲を拡大して考える。

聴覚障害者は、難聴であることが日常生活に与える影響から、「自身が難聴であることを周囲に知られたくない」「知的障害と誤解されたくない」といった心理的負担を抱えることがある。その結果、難聴者としての配慮が行われない場合も多く、他者からの呼びかけに気づけないことで日常的な会話の中で孤立感を深めることが課題となっている。このような状況を改善するためには、聴覚障害者が日常的に利用可能な情報支援ツールや補助装置の導入が必要不可欠である。

（※文責：北清良菜）

1.2　目的

本研究プロジェクトは、聴覚障害者が日常生活において直面する情報へのアクセスの困難や、他者とのコミュニケーションにおける問題、特に呼びかけや公共アナウンスに気づけないことによる孤立感や誤解の発生といった課題を「気づく」ことを可能にすることで解決することを目的とする。

具体的には、既存の支援ツールを参考に、新たな支援システムデバイスの開発および実装を行うことである。本デバイスでは、音声認識エンジンを活用し、事前に登録した特定の言葉（例：装着者の名前）を検出する機能を備える。また、検出時に振動による通知を行うことで装着者に呼びかけを知らせ、「自身で気づいた」という状態を作り出すことを目指す。このようなデバイスは、聴覚障害者が日常生活で情報を円滑に受け取り、コミュニケーションの質を向上させるための有効な手段となると考えられる。

（※文責：北清良菜）

1.3　研究動機

聴覚障害者は日常生活において多くの困難に直面している。特に人とのコミュニケーションにおいて、「呼びかけに反応できない」「孤独を感じる」といった問題が挙げられ、これらは彼らの社会参加や心理的健康に大きな影響を及ぼす。呼びかけに反応できないことは、聴覚障害者が周囲の人々との日常的なやり取りにおいて迅速な対応や情報の共有が困難であることを意味し、この問題は職場や学校、病院など、あらゆる場面で発生する。その結果、コミュニケーションの質が低下し、さらなる孤立を招く要因となっている。

従来、この問題に対する改善策として補聴器の着用が提案されてきた。しかし、補聴器は健常者にとって違和感のあるものと認識されやすく、そのため、補聴器の着用者は「難聴者である」と公然と示しているように捉えられる場合が多い。特に中途失聴者を含む聴覚障害者の中には、自身が難聴であることを認めたくないという心理を持つ者も少なくない。このため、他者から聴覚障害者であると悟られることを避けたいと考える傾向が見られる。

さらに、難聴という状態が老齢や知的障害と誤解されることもあり、それが不快感やストレスを引き起こす要因となる事例も報告されている。このような背景から、他者から聴覚障害者であることを悟られないようなデバイスの開発が求められている。そのデバイスには、着用時に外観が自然で目立たないデザインであること、そして日常生活において支障をきたさない機能性が必要とされる。このようなデバイスは、聴覚障害者が社会生活をより円滑に送ることを支援し、コミュニケーションの質を向上させるとともに、心理的負担を軽減する可能性を有していると考えられる。

（※文責：北清良菜）

1.4　先行研究

本プロジェクトの先行研究として、公立はこだて未来大学卒業生である本多達也氏が開発した聴覚障害者向けデバイス「Ontenna（オンテナ）」を挙げる。Ontenna は音を振動と光に変換することで、ユーザーが音の存在や音量の変化をリアルタイムで感じ取ることを可能にするデバイスである。具体的には、Ontenna は音を感知し、その強さやパターンに応じた振動と光によるフィードバックを提供する。これにより、聴覚障害者は環境音や会話の音を触覚的に感じ取りやすくなる。

また、装着のしやすさも考慮されており、Ontenna はクリップ型のデバイスとして設計されている。このデザインにより、髪や衣服に簡単に取り付けることができ、日常生活の中で自然に使用することが可能である。さらに、充電式バッテリーを搭載しており、一度の充電で長時間の使用が可能である。Ontenna は日常生活のさまざまな場面での使用が想定されており、特に騒音の多い環境や音が重要な情報源となる場面（例：交通信号の音）において有効である。このデバイスは、聴覚障害者が音に関する情報を得る手助けをし、自立した生活を支援することを目的としている。ま

た、健聴者との音の体験を共有することで、相互理解を深める役割も果たしている。このように、音の情報を視覚や触覚など他の感覚を介して伝える取り組みは、聴覚障害者の生活の質を向上させることに寄与している。

本研究プロジェクトでは、聴覚障害者、特に感音難聴者を対象とした新しいデバイスの開発を行う。本研究で開発するデバイスは、Ontenna と同様に音を振動で感じ取ることができるが、その目的や用途において異なる特徴を持つ。

本デバイスの主な機能は、事前に登録した音や言葉を認識し、これを検出した際に振動で装着者に知らせる点である。具体的には、ユーザーが重要と考える音や言葉を登録し、それが環境中で検出されると、デバイスがリアルタイムで振動によるフィードバックを提供する。この機能により、ユーザーは重要な情報や危険を見逃すことなく把握することが可能である。さらに、振動の種類を区別することによって、音の重要度や危険度を伝えることができる。たとえば、軽い振動は日常的な通知を示し、強い振動は緊急性の高い状況を示すように設定することができる。この仕組みにより、ユーザーは振動のパターンから状況の重要性を即座に理解することが可能である。

Ontenna が音の存在や音量の変化を感じ取ることを主眼としているのに対し、本研究で開発するデバイスは、特定の言葉への反応を重視する点で異なる。本研究では、よりパーソナライズされた音情報の伝達を目指しており、用途や目的において先行研究である Ontenna とは明確に区別される。[2][5]

（※文責：北清良菜）

第2章　関連研究

2.1　必要なスキル

本研究プロジェクトを遂行するにあたり、以下のスキルが必要であると考えられる。

- プログラミングスキル（Python）
- 音声認識
- ハードウェア設計

（※文責：櫻田空大）

2.2　解決手法

課題を解決するには、聴覚障害者が聴覚以外の方法で呼びかけに気づけるようにする必要がある。具体的な方法は、事前に名前を登録し、登録された名前が呼ばれたとき、呼びかけに反応して装着者に振動で通知するデバイスを開発することである。

（※文責：櫻田空大）

2.3　関連性の高い専門科目

- オペレーションズリサーチ（特徴量）
- 応用数学 I（フーリエ変換）
- システムプログラミング（ソケット通信）

プロジェクト学習遂行にあたって関連性の高い本学の専門科目は 3 つある。１つ目は応用数学 I である。この科目ではフーリエ変換を学ぶ。音声認識では、周波数を解析するためにフーリエ変換の知識が必要である。２つ目はオペレーションズリサーチである。この科目では特徴量の計算につ

いて学ぶ。フーリエ変換で得られたデータはそのまま処理することが難しく、音声認識の処理に適したデータに変換するために、特徴量の計算が必要である。3 つ目はシステムプログラミングである。この科目ではソケット通信について学ぶ。音声認識で得たデータを即時に処理・応答するために、ソケット通信についての知識が必要である。

（※文責：櫻田空大）

第3章　プロジェクト学習の目標

3.1　最終的な目標

本研究プロジェクト学習の最終的な目標は、聴覚障害者が日常生活において他者からの呼びかけに気づくことができるよう、音声認識技術を活用した振動通知デバイスを開発することである。このデバイスは、聴覚障害者が自分の名前を音声認識エンジンを用いて事前に登録し、その呼びかけが環境中で検出された際に、振動によって通知する仕組みを提供する。これにより、聴覚障害者は音声に頼らず、触覚による情報を得ることができ、日常生活の中で他者とのコミュニケーションをスムーズに行えるようになる。

具体的には、次のような目標を設定する：

1. 音声認識システムの構築： 事前に登録されたキーワード（例：ユーザーの名前）を認識するための音声認識システムを Python で開発する。このシステムでは、録音された音声データをリアルタイムで処理し、特定のキーワードを検出する機能を持たせる。

2. 振動通知機能の実装： Raspberry Pi を使用して、音声認識システムから得られた情報に基づいて、ユーザーに振動通知を送る装置を設計する。

3. デバイスの小型化と自然なデザイン： デバイスの外観は目立たないデザインとし、ユーザーが日常生活で自然に使用できるようにする。特に、ユーザーが聴覚障がいを隠したいと考える場合でも、違和感なく装着できることを重視する。

4. ユーザーインターフェースの改善： ユーザーが簡単に操作できるインターフェースを提供し、名前やキーワードの登録を簡易的に行えるようにする。

最終的には、音声認識と振動通知の精度や信頼性を高め、聴覚障害者の生活の質を向上させるとともに、社会的な孤立を減少させることが目標だ。このデバイスが広く普及することで、聴覚障害者がより自立した生活を送る手助けとなることを期待する。

（※文責：北清良菜）

3.2　アイデア＜醸成＞ワークショップ

本研究プロジェクトは、聴覚障害者が直面する情報アクセスやコミュニケーションにおける問題を解決することを目的としている。従来は、「呼びかけへの迅速な反応」や「孤独感の軽減」を重視して設計を進めていたが、議論を重ねた結果、以下のような変更点を加えた新たな方向性を設定した。

一つ目は、対象者の範囲を拡大した点である。従来は聴覚障害者を主な対象としていたが、本研究では「日常生活で聞き取りが難しいと感じる人々」を含む、高齢者、中途失聴者、軽度難聴者などを新たな対象者として想定することで、より広範な課題に対応できるデバイスを目指すことにした。

二つ目は、目標の再定義を行った点である。従来は「特定の言葉を聞き取ること」を目指していたが、現在では「特定の言葉やアナウンスが発されたことに気づく」ことを重視する。この変更により、音声認識精度への依存を軽減し、音声の存在自体を迅速に検知して利用者に伝える仕組みを構築する。

具体的には、病院や銀行などでのアナウンスに気づくことで情報アクセスを改善し、呼びかけに気づけないことで誤解される事態を防ぐことで、コミュニケーションの質を向上させることを目指す。また、心理的負担を軽減するために、アクセサリーとしても利用可能なネックレス型デバイスを採用し、日常生活に自然に取り入れられるデザインと機能性を追求する。

本研究では、このような変更点を反映した支援システムデバイスの開発および実装を通じて、対象者の生活の質を向上させることを目指す。

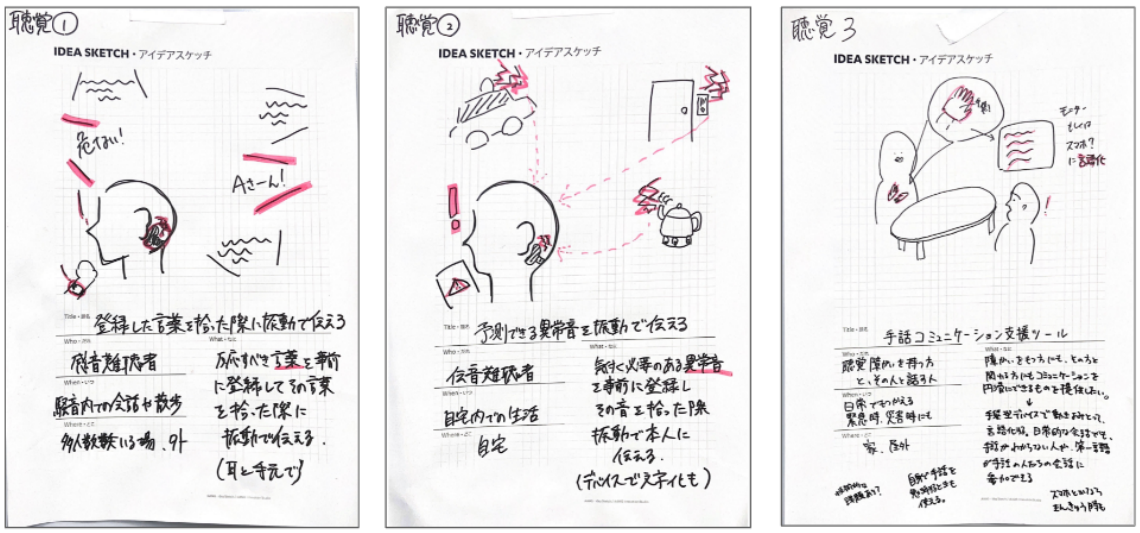


図 3.1: アイデアスケッチ

（※文責：北清良菜）

第4章　目的を達成するための手法、手段

4.1　考察したアイデア

4.1.1　音声認識システムの構築

音声認識システムの構築をするために、オープンソースソフトウェアの活用を検討した。このデバイスはセキュリティ上の理由から、オフラインで動作するものを前提としているため、オフラインで使える音声認識エンジンが必要であると考えた。

（※文責：櫻田空大）

4.1.2　振動通知機能の実装

Raspberry Pi を用いてユーザーに振動で通知を送る方法として、Raspberry Pi に振動モーターを直接つないで操作する方法と、無線の振動モーターを Bluetooth で接続して操作する方法の 2 つを考えた。前者では GPIO ピンの ON/OFF で振動モーターを制御し、後者では Bluetooth 通信を用いて Raspberry Pi と振動モーターを接続する。

（※文責：櫻田空大）

4.1.3　デバイスデザイン

本研究では、聴覚障害者に向けた音の検知と通知を行うデバイスのデザインとして、以下の 3 種類を提案する。それぞれのデザインについて、特徴、利点、欠点を以下にまとめる。

イヤホン型

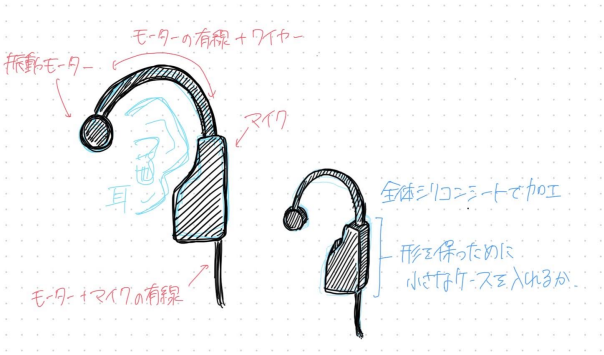


図 4.1: イヤホン型デバイスのデザイン

- **着用箇所**: 耳
- **素材**: シリコンやゴム、ワイヤー、シリコンシート、ケース（プラスチック等）

本デザインは補聴器や骨伝導イヤホンを基にした形状であり、耳の前方に振動モーターを、耳の裏側にマイクを設置する。有線および無線の両方式が組み込み可能である。

メリット

- 既存のデバイスに近い形状のため、新しい技術への不安感が軽減される。
- 耳元に着用することで、音をより健常者に近い形で感知できる。
- デザイン次第ではアクセサリーとして見せることが可能であり、社会的違和感を軽減する。

デメリット

- デバイスがイヤホンに見える反面、補聴器にも見えるため、使用者に心理的抵抗を与える可能性がある。
- イヤーカフに似た原理で装着するため、耳に多少の負担がかかる。
- 有線の場合、耳元からコードを本体まで長く伸ばす必要があるため、取り回しが不便になる。

バンド型

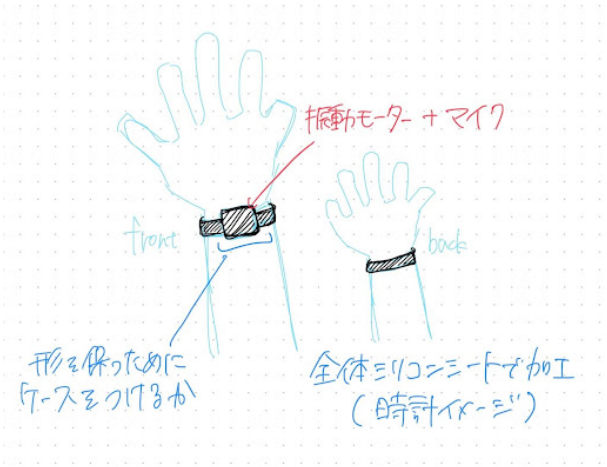


図 4.2: バンド型デバイスのデザイン

- 着用箇所: 手首
- 素材: シリコンやゴム、シリコンシート

本デザインはアップルウォッチやスマートウォッチの形状を基にしており、時計の文字盤部分に振動モーターとマイクを組み込む。無線方式のみが採用可能である。

メリット

- スマートウォッチに似た形状のため、既存技術に近い安心感を持たせることができる。
- 外見が時計やアクセサリーとして認識されやすく、目立ちにくい。
- デザイン次第でファッション性を高めることができる。

デメリット

- 使用するマイクやモーターがサイズに制約を与えるため、無線方式に限定される。
- 手元での作業音が大きい場合、周囲の音声を正確に感知しづらい可能性がある。

ネックレス型

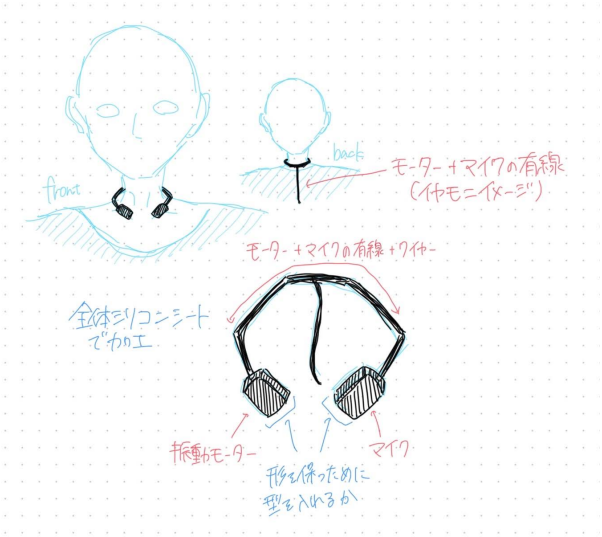


図 4.3: ネックレス型デバイスのデザイン

- **着用箇所**: 首の付け根から鎖骨にかけて
- **素材**: シリコンやゴム、ワイヤー、シリコンシート、ケース（プラスチック等）

本デザインは磁気ネックレスや両耳一体型骨伝導イヤホンを参考にした形状であり、左右の鎖骨部分に振動モーターとマイクを配置する。有線および無線の両方式が選択可能である。

メリット

- 外見が磁気ネックレスに似ており、デザイン次第ではアクセサリーとして見せることが可能である。
- ワイヤーを組み込むことで、首の形や太さに応じた簡単な調節や着脱が可能である。
- 顔に比較的近い位置に装着するため、音を健常者に近い形で感知できる。

デメリット

- 有線の場合、首元からコードを本体まで長く引き伸ばす必要があり、取り回しが不便になる。
- ネックレスやチョーカーを装着することに抵抗を持つ使用者には不向きである。

（※文責：北清良菜）

4.1.4 ユーザーインターフェースの改善

本プロジェクトでは、聴覚障害者が特定の名前や単語に反応できるデバイスの開発を目指した。このデバイスは、ユーザーが反応させたい名前を事前に登録する仕組みを必要とする。そこで、ユーザーが直感的かつ簡単に名前を登録できるようにするため、誰もが使いやすい Web アプリケーションを開発する必要性を考察し、この実現に向けて設計を行った。

（※文責：佐々木大聖）

4.2 新しい解決方法・手段

4.2.1 音声認識システムの構築

オフラインで動作する日本語の音声認識エンジンを調べた結果、VOSK と Julius という 2 つのオープンソースソフトウェアが見つかった。当初は VOSK を使用して開発を進めていたが、名前の登録が難しく実装が困難だったため、本研究プロジェクトでは Julius を採用して開発を行った。

具体的な手法を以下に示す。

1. Julius をモジュールモードで起動
2. モジュールモードで起動した Julius と Python をソケット通信を用いて接続
3. Julius から音声認識の結果を文字列で受け取り、文字列の中に登録されている名前があった場合、それを検出する

なお、Julius と Python を接続する方法については、以下のウェブサイトを参考にした：

「Python で Julius を動かして音声認識システム作ってみた！」
(`https://nekonogorogoro.com/julius_setup/`)

この手順を使用し、マイクから入力された音声を Julius で文字列に変換し、Python を用いて登録された名前を検出する機能を実現した。

（※文責：櫻田空大）

4.2.2 振動通知機能の実装

本研究プロジェクトでは Raspberry Pi に振動モーターを直接接続する方法を採用した。なぜなら、無線の振動モーターを使用する場合、バッテリーの充電管理が必要となり、使いにくいと考えたからである。

具体的な手法としては、4.2.1 で述べた Python プログラムで文字列から名前を検出した際に、振動モーターが接続された GPIO ピンを ON にし、１秒後に OFF にする仕組みを作成した。これにより、登録された名前が呼ばれた場合、ユーザーに１秒間の振動で通知を送ることが可能になった。

（※文責：櫻田空大）

4.2.3　デバイスデザイン

本研究プロジェクトでは、提示した問題解決に応えるため、振動モーターとマイクを用いた聴覚補助デバイスのデザインを検討してきた。その結果、最終的に「ネックレス型」のデザインを採用することとした。以下にその趣旨と採用の根拠を示す。

趣旨

ネックレス型デザインは、日常生活に溶け込みやすいアクセサリー風の外観を持ちつつ、使用者が音を効率よく感知できる機能性を兼ね備えている。首元にフィットするような形状を維持させることで、衣服の下に忍ばせることも可能になる。また、首元という装着箇所は、振動モーターの感触を明確に伝えるのに適しており、マイクの音声入力も健常者の聴覚感覚に近づけることが可能である。このデザインを採用することで、他者利用者ともに違和感を抱くことなく自然にデバイスを使用できるのではないかと考えている。

採用の根拠

1. デザイン性と日常への適応性

   ネックレス型は、外見が磁気ネックレスやアクセサリーに似ているため、聴覚補助デバイスであることを目立たせず、デザイン次第でおしゃれなアクセサリーとしても見せることができる。よって利用者の心理的抵抗感を軽減し、デバイスをより気軽に着用してもらえる可能性がある。

2. 調節性と装着の簡便性

   ネックレス部分にワイヤーを組み込むことで、首の形状やサイズに応じて調節が可能になる。これにより、さまざまな体格の利用者に対応できる柔軟性を備えることができるため、衣服の下に装着することも可能になる。また、簡単な装着と取り外しが可能で、利便性が高い。

3. 音の感知能力の向上

   首の付け根から鎖骨にかけて装着することで、マイクを顔に比較的近い位置に配置できる。これにより、健常者の聴覚感覚に近い形で音を拾うことができ、音声認識の精度を向上させることが期待できる。

4. 振動の伝達効率

振動モーターを左右の鎖骨部分に配置することで、利用者に振動を効率よく伝えられる。この位置は振動感知に適しており、デバイスの通知機能がより確実に伝達されることができる期待が高い。

デメリットへの配慮

有線仕様におけるコードの取り回しや、ネックレス型デバイスへの抵抗感といったデメリットについても認識している。これらを補うために、無線方式の選択肢を加えるとともに、利用者の多様な好みに応じたデザインや素材のバリエーションを用意することで緩和する。

以上の理由から、4.1.3 であげた案の中で最もネックレス型デザインは実用性、快適性、デザイン性を兼ね備えた最適な選択肢であると判断した。このデバイスを通じて、聴覚補助を必要とする方々の日常生活をより豊かにすることを目指す。

（※文責：北清良菜）

4.2.4　ユーザーインターフェースの改善

音声認識ツール「Julius」を活用して特定の名前を検出するには、認識対象となる単語を互換性のある音素表現に変換する必要がある。これを効率的に実現するために、以下の方法を採用した：

1. ユーザーが Web アプリケーション上で名前を入力すると、バックエンドで音素表現に変換するプロセスを構築。
2. この変換データを基に音声認識エンジンの辞書ファイルを自動更新する仕組みを導入。
3. 音声認識システムを再起動し、更新された辞書ファイルを再読み込みすることで、新しい名前がすぐに認識可能になるよう設計した。

この一連のフローにより、ユーザーは複雑な操作を必要とせず、簡単に任意の単語や名前を登録できるようになった。

（※文責：佐々木大聖）

4.3　用いる技術

4.3.1　音声認識システムの構築

音声認識システムの構築には以下の技術を使用した。

- 音声認識エンジンの操作
  本研究プロジェクトでは、音声認識エンジンとして Julius を使用した。Julius をモジュールモードで起動することで、Python で音声データを処理できる環境を構築した。

- ソケット通信

  Julius をモジュールモードで起動させ、そこで得た音声認識結果を Python で処理するために、Julius と Python をソケット通信で通信させた。

- データ解析

  ソケット通信を通じて Julius が出力する音声認識結果を Python で受信し、その中から登録された単語を抽出する。正規表現を使用して、文字列から登録した単語の有無を解析した。

（※文責：櫻田空大）

4.3.2 振動通知機能の実装

振動通知機能の実装には以下の技術を使用した。

- GPIO ピン制御

  音声認識システムで登録された単語があった場合、Raspberry Pi の GPIO ピンを使用して振動モーターを制御する。具体的には、振動モーターを接続した GPIO ピンを HIGH (ON) に設定してモーターを動作させ、１秒後に LOW (OFF) に設定することで振動モーターを停止させる。これにより、振動で通知を送る機能を実現する。

（※文責：櫻田空大）

4.3.3 デバイスデザイン

小型化・軽量化技術

首元で快適に装着可能なデバイスにするため、軽量でコンパクトな設計が必要であると考えた。

- 実装概要

  シリコンやゴム素材を使用し、軽量で柔軟性のあるデザインを実現。

（※文責：北清良菜）

電子回路設計における工夫

Raspberry Pi 5 を用いて振動モジュールを制御する回路を設計・構築した。設計および組み立てにおいて、小型デバイス特有の課題である「はんだ付けの強度確保」および「回路間の短絡防止」に対し、接続部分をシリコンチューブで覆う処理を行い、接触防止・振動吸収を行った。

（※文責：佐々木大聖）

4.3.4　ユーザーインターフェースの改善

本研究プロジェクトでは、ユーザーが簡単に名前や単語を登録できる Web アプリケーションを開発し、音声認識システムの柔軟性を向上させた。この Web アプリケーションでは以下の技術を使用して実装を行った。

Flask を用いたローカルサーバーの構築

Web アプリケーションの基盤として Flask を使用し、ローカルサーバーを構築した。これにより、ユーザーがアプリケーションを通じて送信した名前のデータをバックエンドで処理する仕組みを実現した。

音素ファイルと辞書ファイルの自動更新

Web アプリケーションから送信された名前データを音素表現に変換し、音声認識エンジン「Julius」の辞書ファイルを自動的に書き換えるプロセスを実装した。この仕組みにより、ユーザーは任意の名前を登録し、音声認識システムにリアルタイムで反映させることが可能となった。

音声認識システムの再起動

辞書ファイルの更新後に音声認識システムを自動的に再起動するプロセスを組み込み、新しい名前や単語の登録を即座に反映できるよう工夫した。

これらの技術的な工夫により、ユーザーが反応させたい名前や単語を直感的かつ柔軟に設定できるシステムを構築することができた。また、Web アプリケーションによって登録された名前は、以下のように音素表現に変換され、システム内で使用された。

bccwj.60k.htkdic ファイル

たろう+名詞 [たろう]　ｔａｒｏｕ

bccwj.60k.pdp.htkdic ファイル

たろう+名詞 [たろう]　t_B a_I r_I o_I u_E

trigger_words.txt ファイル

たろう

これらの仕組みは、ユーザーが登録した名前が音声認識エンジン「Julius」に正確に反映され、必要な単語や名前を効率的に認識できる環境を提供した。これにより、システム全体のユーザビリティが大幅に向上した。

実際の Web アプリケーションの画面については、以下の図に示す。

（※文責：佐々木大聖）

(a) 名前を追加する画面

(b) 名前リストの画面

図 4.4: Web アプリケーションの UI

4.4　技術構成

本システムは、Web アプリケーション、音声認識エンジン「Julius」、Python スクリプト、Raspberry Pi 5、および振動モーターによって構成されている。

ユーザーが Web アプリケーションを通じて名前を登録すると、Flask によるローカルサーバーがその情報を受信し、「Julius」の辞書ファイルを自動的に更新する。音声認識エンジンは USB マイクから音声を取得し、指定された名前を検知した場合、Raspberry Pi を介して振動モーターに信号を送信する。これにより、呼びかけを検知して利用者に通知する仕組みを実現した。

本システムの技術構成を以下の図に示す。

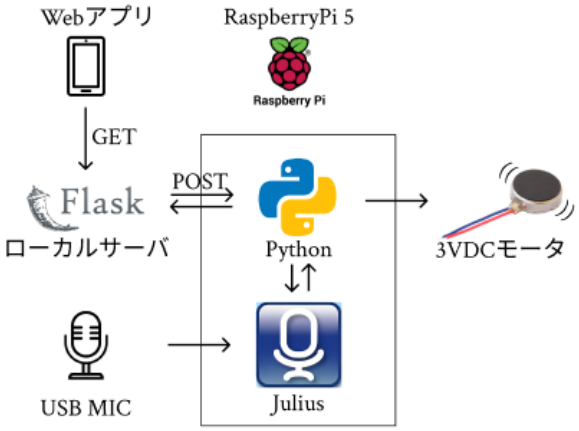


図 4.5: 技術構成図

（※文責：佐々木大聖）

第5章　結果

5.1　聴覚支援グループの成果

本研究プロジェクトでは、聴覚障害者が日常生活において直面する情報へのアクセスの困難や、他者とのコミュニケーションにおける問題、特に呼びかけや公共アナウンスに気づけないことによる孤立感や誤解の発生といった課題を「気づく」ことを可能にすることで解決することを目的として活動を行ってきた。この活動を通して、実際に自分への呼びかけに気づくことができるデバイスを開発することができた。

開発したデバイスは、音声認識エンジン「Julius」を活用し、ユーザーが事前に登録した名前や特定の言葉をリアルタイムで検出し、それを振動で通知する仕組みを備えている。この機能により、聴覚障害者が他者からの呼びかけに気づきやすくなり、結果としてコミュニケーションのきっかけを生み出すことが可能となった。また、音声認識の柔軟性を高めるため、名前や単語の登録を簡単に行える Web アプリケーションも開発し、デバイスの操作性を向上させた。

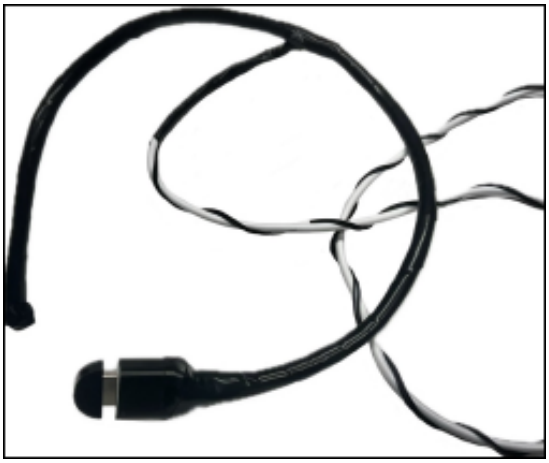
さらに、デバイスのデザインにも注力し、イヤホン型、バンド型、ネックレス型の 3 つの案を検討した結果、アクセサリーとしても使用可能なネックレス型デザインを採用した。このデザインは、装着時に他者から気づかれにくいだけでなく、自然に日常生活に溶け込む形状とし、心理的負担を軽減することを目指した。首元に装着することで振動の感触が伝わりやすく、音声通知の効果を最大化する設計を実現した。

プロトタイプの評価では、日常生活のシミュレーションを通じて、デバイスが効果的に動作することが確認された。特に、周囲の呼びかけに気づくことが困難な状況で、このデバイスが情報を受け取る支援を提供できることが明らかになった。

本プロジェクトの成果として、デバイスの開発とその技術的な実現に加え、聴覚障害者のニーズを深く理解し、それに応える製品を設計・評価するプロセスを確立した点が挙げられる。このデバイスが広く利用されることで、対象者がより円滑な社会参加を果たし、生活の質を向上させることが期待される。

本研究プロジェクトで開発した成果物の図と装着時の例を以下に示す。

この図では、服の上から装着した状態を示しているが、実際の使用時には服の下に装着することを想定している。



(a) 成果物：名前に反応するデバイス

(b) デバイス装着例（服の上から装着した状態）

図 5.1: 成果物

（※文責：佐々木大聖）

5.2 中間成果発表

概要

中間成果発表会ではアンケートフォーラムを用いて、プロジェクト間で相互評価を行った。評価フォームには「発表技術についての評価」と、「発表内容についての評価」をそれぞれ 10 段階で評価し、コメントを記入する項目を設けた。それぞれの評価の基準は、「プロジェクトの目的と計画を伝えるために効果的な発表が行われているか」と、「プロジェクトの目標設定と計画は適切か」であった。

評価

本プロジェクトでは、23 件のアンケート結果を得た。内訳は学生が 18 名、教員が 5 名であった。全体または聴覚支援グループの発表技術についての評価結果は図 1 のように、平均 8.30 であった。コメントの一部を以下に示す。

- よく練習されていて、資料もアニメーションがあるなど理解しやすい。
- 流れがスムーズで要点がまとめられていて分かりやすかった。
- 聴衆の方を見ながら大きな声でしっかりと発表しており、とても聞きやすかった。
- まとめ方が綺麗で、声が聞こえやすかった。
- デバイスの動作が映像化されていて分かりやすい。聴覚障害者が抱える問題を挙げ、その中で着目した点が分かりやすかった。
- ポスターをもう少し活用してもよかったのでは？

全体または聴覚支援グループの発表内容についての評価結果は図 2 のように、平均 8.13 であった。コメントの一部を以下に示す。

- 課題設定の理由、アプローチの方法が理論立てて簡潔に、さらに写真や動画を用いて説明されていてとても伝わりやすかった。
- 目標、展望が共に具体的でいいと思いました。
- 目的や目標は分かりやすかったが、なぜこうしたかの理由はもっと細かくてもいいかと思った。
- 聴覚障害の話はありがちな印象で、キーアイデアがあまりわからなかった。
- 聴覚支援班：どの方向から呼ばれたかとかはわかるような実装でしょうか？

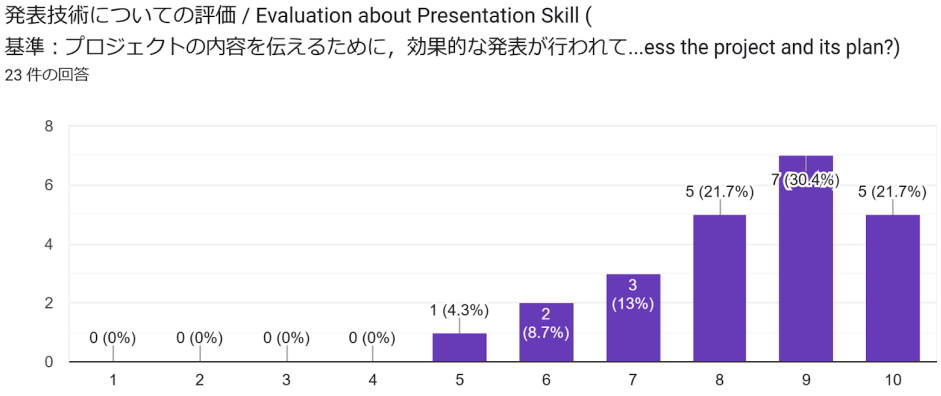


図 5.2: 中間: 成果発表技術の評価

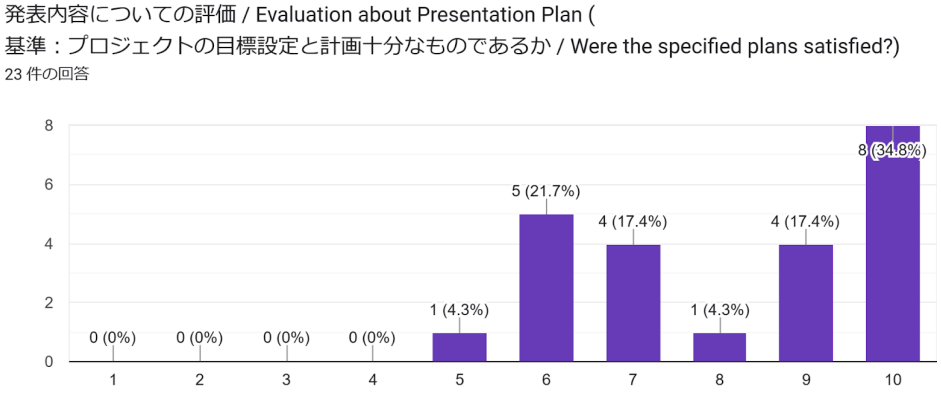


図 5.3: 中間: 成果発表内容の評価

評価を踏まえた反省

中間発表会において、発表技術については全体的に高評価であった。プレゼンテーションの構成や資料、レイアウトが見やすいという評価が多く、特に動画を使ったデバイスの説明が分かりやすかったという意見が多かった。声がハキハキしていて聞き取りやすかったという評価も多く、早い段階からの準備が評価に繋がったと考えている。発表内容については、簡潔なプレゼンをしようと心掛けた結果、説明不足になってしまった点があった。デバイスに関しては実用的であるという評価が多かった一方で、革新性の低さや、必要性についての指摘があった。

（※文責：佐々木大聖）

5.3　期末成果発表

概要

期末成果発表会においても、アンケートフォーラムを用いて、プロジェクト間で相互評価を行った。評価フォー ムには「発表技術についての評価」と、「発表内容についての評価」をそれぞれ 10 段階で評価し、コメントを記入する項目を設けた。それぞれの評価の基準は、「プロジェクトの目的と計画を伝えるために効果的な発表が行われているか」と、「プロジェクトの目標設定と計画は適切か」であった。

評価

本プロジェクトでは、23 件のアンケート結果を得た。内訳は学生が 22 名、教員が 1 名であった。全体または聴覚支援グループの発表技術についての評価結果は図 1 のように、平均 8.13 であった。コメントの一部を以下に示す。

- 聞きやすい声とスピードだった。
- スライドが見やすくて、ポスターがわかりやすくて、声聞きやすかった。
- とても聞きやすく、順序立てて説明していたので良かった。PV も簡潔で分かりやすい。
- グループごとに取り組みが違っても、発表スライドのまとめ方などグループ全体で連携が取れているのが伝わってきた。
- わかりやすい説明でした。しかし、ところどころ何を強調したいかわからない部分がありました。
- 時間が結構過ぎていたので発表練習を時間を計ってやって練習したほうがいいと思った。
- せっかくポスターを並べたのだからもう少し活用したり、作ったデバイスでデモとかがあると良かった

全体または聴覚支援グループの発表内容についての評価結果は図 2 のように、平均 8.87 であった。コメントの一部を以下に示す。

- 課題発見、解決、今後の展望がしっかりしていた。
- 目的と活動の繋がりのが明確で非常に良かった。
- 問題解決がしっかりできていそうなデバイスが開発されていて良かった。
- 課題に対する解決策がとても良いなと思った。
- デバイスのデザインも、実際に使用する場面を想定して作られていてとても良いと思った。

- 聴覚障害者の視点に立ち、他人に気づかれにくいデバイスにしたというところが思いやりがあっていいと思った。服の中に入れると音声を拾いにくい状態になるが、その状態でも正確にデバイスが作動するのかが争点だと思う。
- 名前の判定はどうやって？(名字だけとか、名前だけとか、あだ名とかは？)
- 聴覚支援について、ユースケースが示されていてわかりやすかった。

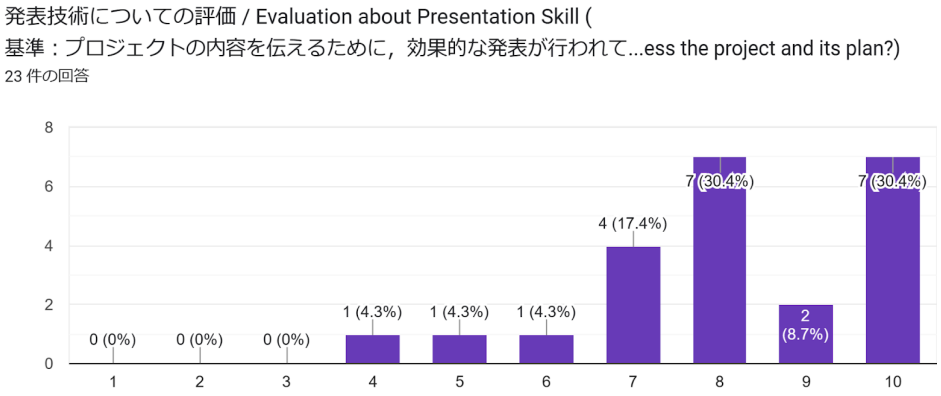


図 5.4: 期末: 成果発表技術の評価

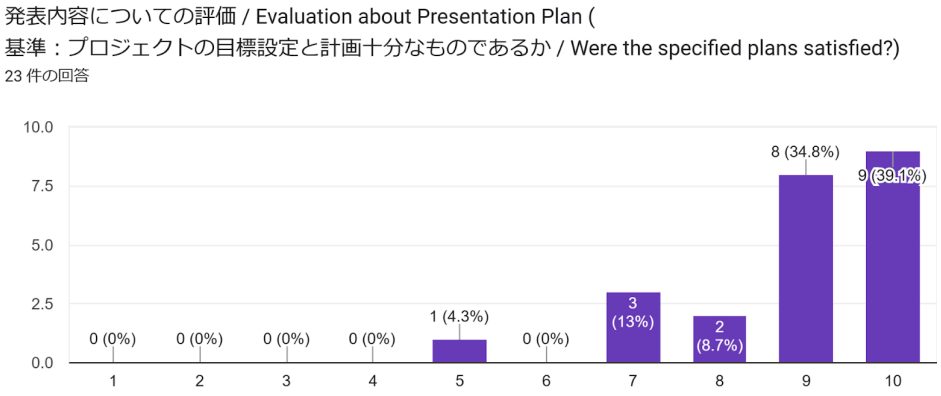


図 5.5: 期末: 成果発表内容の評価

評価を踏まえた反省

期末発表会では、発表技術については全体的に高評価を得ることができたが、平均評価は中間発表の結果よりもわずかに下回る結果となった。主な要因として、発表時間内で間に合わない回があったことが挙げられる。このようなアクシデントが発生しても柔軟に対応できるよう、発表練習を時間を計りながら繰り返し行っておくべきだったと感じた。一方で、スライドやポスターが見やすく、声が聞き取りやすいという点は好評を得ており、「どのような活動を行ったか」をしっかり伝えられた点は良い成果といえる。発表内容については、中間発表よりも高い評価を得ることができ

た。これは、課題設定から解決方法、今後の展望に至るまで一貫性を持って活動を進めていたことが評価に繋がったと考えられる。また、自分たちで立てた目標に向かって着実に活動を行い、それを効果的に伝えることができたことが好評価の理由として挙げられる。

（※文責：佐々木大聖）

5.4　学外アドバイザーとの意見交換

本研究プロジェクトでは、島影圭佑准教授の指導のもと、学外アドバイザーとの意見交換を通じて、デバイスの設計やデザインについて多くの知見を得た。具体的には、大学の非常勤講師を務めている須田拓也氏および株式会社 STYZ インクルーシブデザインスタジオ CULUMU CDO 川合俊輔氏とのディスカッションを行い、デザインとユーザー体験に関する貴重なアドバイスを受けた。

デザインに関しては、使用シーンを基に考察することで、より良い案を導き出す手法を学んだ。また、発表時には、このシーンにおける必要性を明確に示すことで、説明の説得力を高める重要性も理解した。デバイス選定においては、音声認識に適した Raspberry Pi の選択方法や、ユーザーとデバイスの関係性について考える契機となり、開発方針を再確認することができた。

特に、開発が遅れている最中でのアドバイスは、メンバー全員で方針を共有し、プロジェクトを進展させる上で大変有意義であった。また、デバイスが完成していなくても、その開発過程から多くの学びが得られることを認識した。これらの経験は、プロジェクト全体の方向性と成果に大きく貢献した。

（※文責：佐々木大聖）

5.5　NT 名古屋の振り返り

2024 年 11 月に参加した NT 名古屋では、聴覚支援班が開発したデバイスを展示し、多くの来場者に興味を持っていただく機会となった。特に、お年寄りの来場者からは「これがあったら便利だねぇ」との声をいただき、デバイスの実用性や社会的な需要を強く実感する場となった。

一方で、イベント会場の環境が非常に騒がしく、周囲の音が大きかったため、音声認識機能が十分に発揮されず、デバイスのデモを思うように行うことができなかった。また、プログラムの仕様により、一度動作すると終了してしまうという課題があり、継続的な動作が困難であったことも大きな改善点として認識された。

このようなアウトプットの場では、デバイスの実用性を確認するとともに、改良すべき課題を発見できるため、非常に意義があると感じた。今後の研究活動においても、このように多くの人々から意見をもらえる場を積極的に設け、デバイスの完成度を高めることを目指していきたい。

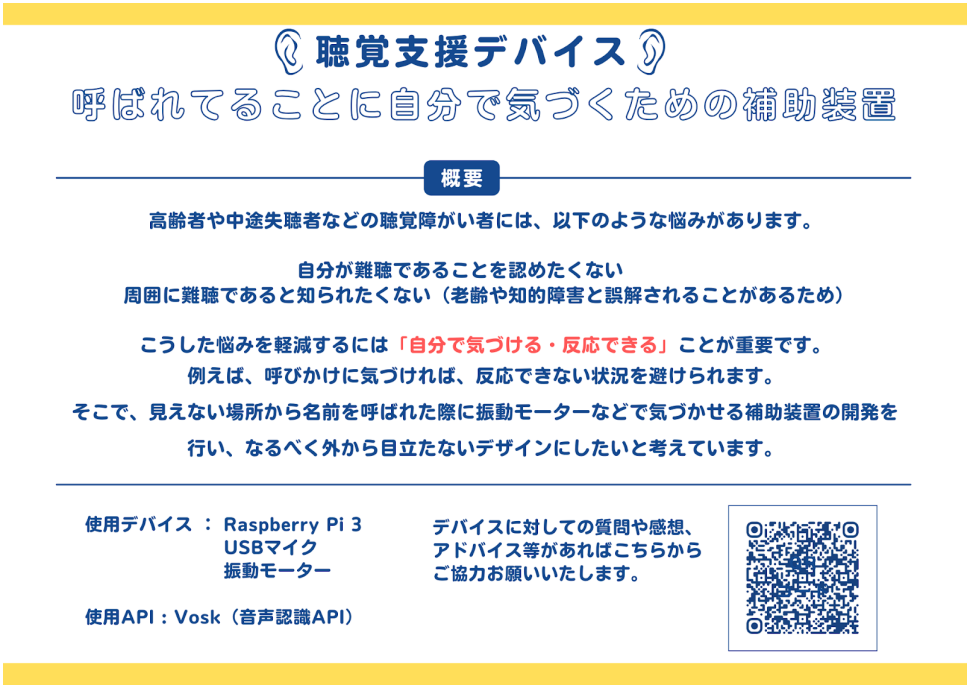


図 5.6: NT 名古屋で使用した資料

（※文責：佐々木大聖）

第6章　考察

6.1　得られた成果

本研究プロジェクトで開発したデバイスにより、聴覚障害者が情報の取りこぼしを減らし、円滑なコミュニケーションのきっかけを得られる可能性を示すことができた。また、振動モジュールを活用した通知機能により、呼びかけや重要な音声情報に気づきやすくなるという実用性が確認された。

今後の課題として、このデバイスを実際に聴覚障害者に使用してもらい、操作性や実用性を検証する必要があると考える。また、そのフィードバックを基に改良を重ねることで、デバイスの完成度をさらに向上させたい。

発表スライドやポスターについては、中間発表・期末発表のいずれにおいても好評を得た。特に図やグラフを用いた説明が視覚的に分かりやすかったため、今後もこのような工夫を取り入れることで、より効果的にプロジェクトの成果を伝えられることがわかった。

さらに、振動モジュールの装着部位に関する考察では、振動が最も伝わりやすいのは手先や足先など血管が集中している部位であることがわかった。しかし、これらの部位は日常生活で装着するには不便であるため、見た目に目立たず、気軽に装着できる場所が必要である。その結果、首元が最も適切な装着部位であると考えられる。首元は振動が伝わりやすく、ハンズフリーで利用可能な点で優れており、日常生活においても自然な使用が期待できる。

（※文責：佐々木大聖）

6.2　妥当性

本研究プロジェクトで提案したデバイスの設計および機能は、課題に対して妥当な解決策であると考えられる。デバイスは、音声認識技術を利用してリアルタイムでユーザーに通知を行い、従来の補聴器や情報支援ツールにはないパーソナライズ機能を実現した。さらに、日常生活に自然に溶け込むアクセサリー型のデザインや、ユーザーが簡単に名前を登録できる Web アプリケーションの導入も、現実的な利用シナリオに即していると考える。一方で、試作品の評価は限定的な環境での検証に留まったため、さらなる実証実験が必要である。

（※文責：佐々木大聖）

6.3　課題点

本研究プロジェクトでは、聴覚障害者を支援するデバイスの開発とそのシステム実装を優先的に行ったが、いくつかの課題点が残されている。以下に主な課題と今後の検討事項を示す。

デバイスデザインにおける主な課題と今後の検討事項

1. 無線化の未実現
   本研究プロジェクトでは、デバイスの実装を有線方式で行ったため、動作には問題がないものの、日常生活での使用を想定すると配線が煩雑になるという課題がある。無線モーターや無線マイクを導入することで、利便性を大幅に向上させる可能性があり、システム的には対応が可能であることから、今後の改善点として挙げられる。

2. 素材選定の限定性
   使用した素材について比較検討を行うことができなかったため、使用感や個人の好みに応じた最適な素材の組み合わせや、より良い形状のデザインが見逃されている可能性がある。ユーザーのフィードバックをもとに、さまざまな素材や形状の試作・評価を行う必要がある。

3. 携帯性の課題
   本研究では、身に着けるデバイスの開発に重点を置いたため、システムの中核であるラズベリーパイの携帯性については考慮していない。日常生活での利用を想定すると、ラズベリーパイの持ち運びや収納方法に関する検討も重要であると考えられる。

システム内ソースコードの改善点

1. 音声認識精度の向上
   本研究では、音声認識エンジン「Julius」を使用したが、認識精度のさらなる向上が求められる。ノイズ環境下での精度向上や特定の単語認識の最適化が今後の課題である。

2. 編集の簡易化
   現在、音声認識と Web アプリケーションのコードが同一プログラム内にあるため、編集が難しい。これを解決するために、コードのモジュール化や分割による管理の容易化が必要である。

以上の課題を解決することで、本プロジェクトで開発したデバイスの実用性をさらに向上させ、聴覚障害者がより快適に利用できる支援デバイスへと発展させることが期待できる。

（※文責：北清良菜・佐々木大聖）

6.4　本学との関連性

本研究プロジェクトは、音声認識、ハードウェア設計、ユーザーインターフェース開発といった多分野にまたがる知識と技術を統合した点で、本学のカリキュラムと強く関連している。特に、応用数学やシステムプログラミング、オペレーションズリサーチなどの専門科目で学んだ知識を応用し、音声認識やデバイス設計に役立てた。また、プロジェクト学習の目標である、実社会の課題に向き合い、実践的に解決策を模索する経験を積むことができた。

（※文責：佐々木大聖）

6.5　拡張性

本研究で開発したデバイスは、聴覚障害者を支援するための基礎的な機能を実装しているが、以下のような拡張性を備えており、さらなる改良や用途の拡大が可能である。

1. 無線化による利便性向上デバイスに無線モーターや無線マイクを導入することで、配線の煩雑さを解消し、日常生活での使用性を向上させることができる。Bluetooth や Wi-Fi などの無線通信技術を取り入れることで、よりシームレスな使用が可能になる。

2. 音声認識の高精度化とカスタマイズ現在は特定の単語が発されたことを検知する仕組みを採用しているが、音声認識技術を進化させることで、より複雑なフレーズや環境音の認識が可能になる。

3. 素材と形状の多様化使用者の好みや用途に応じた素材や形状の選択肢を増やすことで、より快適で長時間の利用に適したデザインが可能になる。例えば、防水素材を採用することで屋外やスポーツシーンでの利用が可能になる。

4. スマートデバイスとの連携デバイスをスマートフォンやスマートウォッチと連携させることで、通知内容を視覚的に確認したり、設定を簡単に変更したりできるようになる。また、スマート家電や他の IoT デバイスと接続することで、より広範な支援システムを構築することも可能である。

5. 用途の多様化聴覚障害者以外の利用者にも応用できるように設計を拡張することで、例えば、騒音の多い環境での音声通知や、高齢者へのリマインダー機能を備えたデバイスとして活用できる。

6. 携帯性の向上デバイス本体の小型化や、ラズベリーパイの統合化により、より軽量で携帯性の高いデザインへの進化が期待される。これにより、日常生活での利用がさらに容易になる。

これらの拡張性を活用することで、デバイスの実用性と適用範囲を大幅に広げることができ、利用者の多様なニーズに応えることが可能となると考える。

（※文責：北清良菜）

6.6　今後の展望

本研究プロジェクトで開発したデバイスをより実用的なものにするためには、以下の取り組みが重要であると考える。

- 実際のユーザーからのフィードバックの収集
  私たちが制作したデバイスを実際に聴覚障害者に使用してもらい、実用性や操作性についてのフィードバックを得ることが必要である。このフィードバックを基に改良を重ねることで、デバイスの完成度とユーザー満足度を向上させることができる。

- 公共の場での使用検証
  デバイスを公共の場で使用し、さまざまな環境下での動作を検証する必要がある。具体的には、私生活をはじめとした、駅や商業施設、学校など多様な場面でテストを行い、環境音の影響や動作の安定性を確認する。この結果に基づき、音声認識のアルゴリズムやデバイスの性能をさらに最適化することを目指す。

- 音声認識精度の向上
  音声認識の精度を向上させることで、デバイスのユーザビリティを高めることが可能である。特に、ノイズの多い環境や遠くからの呼びかけにも対応できるよう、認識能力の強化を図る。また、特定の名前や単語の検出に加えて、文脈を理解する能力の向上にも取り組みたいと考えている。

- デバイスの改良と機能拡張
  無線化やデザインの改良により、さらに使いやすいデバイスへと進化させる。また、スマートフォンや他のデバイスとの連携機能を追加し、日常生活のさまざまな場面で役立つ支援ツールとしての可能性を広げる。

- Web アプリの構造改良
  現在のシステムでは、Web アプリについてのコードが音声認識プログラムと同じプログラム内に置かれているため、コードの管理や編集がやや複雑になっている。この点を改善するため、Web アプリを別のページとして読み込む構造に変更し、各モジュールの独立性を高めることで、プログラムの保守性や編集の容易さを向上させる予定である。

（※文責：佐々木大聖）

参考文献

[1] A. Lee, T. Kawahara and K. Shikano. ”Julius - An Open Source Real-Time Large Vocabulary Recognition Engine”. In Proc. EUROSPEECH, pp.1691-1694, 2001.

[2] Fujitsu. (2023). ”感じること、それが未来。Ontenna ＜オンテナ＞”. Ontenna. https://ontenna.jp/

[3] ノマド・ラプソディー. ”Python で Julius を動かして音声認識システム作ってみた！”, (2024/01/20). https://nekonogorogoro.com/julius_setup/

[4] キャサリン・ブートン. (2016). 人生の途上で聴力を失うということ　心のマネジメントから補聴器、人工内耳、最新医療まで. (ニキ　リンコ, Trans.). 明石書店.

[5] 公立はこだて未来大学. (2017.06.14). ”本学卒業生の本多達也さんがグッドデザイン賞を受賞”. 公立はこだて未来大学. https://www.fun.ac.jp/news/2042

[6] 難聴者の心理学的問題を考える会編; 今尾真弓 [ほか執筆]. (2020). 難聴者と中途失聴者の心理学: 聞こえにくさをかかえて生きる. かもがわ出版.

（※文責：北清良菜）

付録

成果発表時に用いたポスター

図 6.1: 聴覚支援班ポスター

（※文責：佐々木大聖）

本文中に用いたプログラムのソースコード

Program 6.1: Server.py

```
from flask import Flask, request, redirect, url_for, render_template
import os
import pykakasi
import shutil
import subprocess
import time
import signal
import socket
import netifaces as ni

def get_wifi_ip(interface='wlan0'):指定された
    """Wi-インターフェースのアドレスを取得 FiIP"""
    try:
        ip_address = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
        return ip_address
    except (KeyError, ValueError, IndexError):
        raise RuntimeError(fインターフェース" '{interface}' のアドレスを取得できませんでした。IP")

# の設定 Julius
JULIUS_DIR = "/home/dlite3/julius/dictation-kit-4.5"
JULIUS_COMMAND = "julius -C main.jconf -C am-gmm.jconf -module -charconv utf-8 sjis"

def start_julius():
    """を起動する関数。
    Juliusで実行し、プロセスの情報を表示する。
    lxterminal
    """
    julius_lxterminal_command = f"lxterminal -e '{JULIUS_COMMAND}'"
    subprocess.Popen(julius_lxterminal_command, shell=True, cwd=JULIUS_DIR)

def stop_julius():
    """起動中のを停止する関数。
    Juliusプロセスを取得してシグナルを送信する。
    IDSIGTERM
    """
    time.sleep(0.5)
    julius_pid = None

    # コマンドでプロセスを検索 ps
    for line in os.popen("ps aux | grep 'julius' | grep -v 'grep'"):
        fields = line.split()
        julius_pid = int(fields[1]) # を取得 PID

    # プロセスを停止
    if julius_pid:
        os.kill(julius_pid, signal.SIGTERM)
    else:
        print("のプロセスが見つかりませんでした。Julius")

#の設定 python
PYTHON_DIR = "/home/dlite3/julius"
PYTHON_COMMAND = "python NameRecoVibe.py"
```

```
53
54 def start_python():
55     """を起動する関数。
56     pythonで実行し、プロセスの情報を表示する。
57     lxterminal
58     """
59     python_lxterminal_command = f"lxterminal -e '{PYTHON_COMMAND}'"
60     subprocess.Popen(python_lxterminal_command, shell=True, cwd=PYTHON_DIR)
61
62 def stop_python():
63     """起動中のを停止する関数。
64     pythonプロセスを取得してシグナルを送信する。
65     IDSIGTERM
66     """
67     time.sleep(0.5)
68     julius_pid = None
69
70     # コマンドでプロセスを検索 ps
71     for line in os.popen("ps aux | grep 'NameRecoVibe' | grep -v 'grep'"):
72         fields = line.split()
73         python_pid = int(fields[1]) # を取得 PID
74
75     # プロセスを停止
76     if python_pid:
77         os.kill(python_pid, signal.SIGTERM) # ソフトな停止
78     else:
79         print("のプロセスが見つかりませんでした。python")
80
81 app = Flask(__name__)
82
83 # の初期化 kakasi
84 kks = pykakasi.kakasi()
85
86 # ファイルパス
87 pdp_dic_file = "/home/dlite3/julius/dictation-kit-4.5/model/lang_m/bccwj.60k.
       pdp.htkdic"
88 simple_dic_file = "/home/dlite3/julius/dictation-kit-4.5/model/lang_m/bccwj.60
       k.htkdic"
89 trigger_file = "/home/dlite3/julius/trigger_words.txt" # トリガーファイルのパス
90
91 # 名前を形式に変換 PDP
92 def format_name_pdp(name):
93     romaji = "".join([result['hepburn'] for result in kks.convert(name)])
94     formatted_romaji = []
95     skip_next = False
96
97     for i, char in enumerate(romaji):
98         if skip_next:
99             skip_next = False
100             continue
101
102         # 子音+をつの音素として扱う h1
103         if i < len(romaji) - 1 and char + romaji[i + 1] in ["sh", "ch"]:
104             combined = char + romaji[i + 1]
105             if i == 0:
106                 formatted_romaji.append(combined + "_B")
107             elif i == len(romaji) - 2:
108                 formatted_romaji.append(combined + "_E")
```

```
109             else:
110                 formatted_romaji.append(combined + "_I")
111             skip_next = True
112         elif char == "o" and i > 0 and romaji[i - 1] == "o": # 連続する"o" は
                長音として":"
113             formatted_romaji[-1] += ":"
114         elif char == "u" and i > 0 and romaji[i - 1] == "u": # 連続する"u" は
                長音として":"
115             formatted_romaji[-1] += ":"
116         elif char == "n" and (i == len(romaji) - 1 or romaji[i + 1] not in "
                aiueo"): # ん"" の場
                合
117             if i == 0:
118                 formatted_romaji.append("N_B")
119             elif i == len(romaji) - 1:
120                 formatted_romaji.append("N_E")
121             else:
122                 formatted_romaji.append("N_I")
123         else:
124             if i == 0:
125                 formatted_romaji.append(char + "_B")
126             elif i == len(romaji) - 1:
127                 formatted_romaji.append(char + "_E")
128             else:
129                 formatted_romaji.append(char + "_I")
130
131     return f"{name名詞}+ [{name}]\t{’ ’.join(formatted_romaji)}"
132
133 # 名前を形式に変換 Simple
134 def format_name_simple(name):
135     romaji = "".join([result[’hepburn’] for result in kks.convert(name)])
136     phonetic_representation = []
137     skip_next = False
138
139     for i, char in enumerate(romaji):
140         if skip_next:
141             skip_next = False
142             continue
143
144         # 子音+をつの音素として扱う h1
145         if i < len(romaji) - 1 and char + romaji[i + 1] in ["sh", "ch", "ry",
                "ny", "ky", "gy", "hy", "my", "py", "by"]:
146             phonetic_representation.append(char + romaji[i + 1])
147             skip_next = True
148         elif i > 0 and char in ["o", "u"] and romaji[i - 1] == char: # 長音に
                対応
149             phonetic_representation[-1] += ":"
150         elif char == "n" and (i == len(romaji) - 1 or romaji[i + 1] not in "
                aiueo"): # ん"" を"N" に変
                換
151             phonetic_representation.append("N")
152         else:
153             phonetic_representation.append(char)
154
155     return f"{name名詞}+ [{name}]\t{’ ’.join(phonetic_representation)}"
156
157 # トリガーファイルの読み取り
158 def load_trigger_words():
159     try:
```

```
        with open(trigger_file, "r", encoding="utf-8") as f:
            return [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        return []

# ファイルからエントリを削除
def delete_entry(name, filename):
    try:
        with open(filename, "r", encoding="utf-8") as f:
            lines = f.readlines()
        with open(filename, "w", encoding="utf-8") as f:
            for line in lines:
                if name not in line.strip():
                    f.write(line)
    except FileNotFoundError:
        pass

# バックアップを作成
def backup_file(filename):
    backup_filename = filename + ".backup"
    shutil.copy(filename, backup_filename)

# 辞書にエントリを追加
def add_entry_to_htkdic(content, name, filename):
    try:
        with open(filename, "r", encoding="utf-8") as f:
            if any(name in line for line in f):
                return fエントリ" '{name}' は既に存在します。"
        backup_file(filename)
        with open(filename, "a", encoding="utf-8") as f:
            f.write("\n" + content)
        return fエントリ" '{name}' を追加しました。"
    except FileNotFoundError:
        return 指定されたファイルが見つかりません。""
    except Exception as e:
        return fエラーが発生しました": {str(e)}"

@app.route('/')
def home():
    return '''
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <title名前を辞書に追加></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center名前を辞書に追加"></h1>
        <form method="POST" action="/confirm" class="mt-4">
            <div class="mb-3">
                <label for="name" class="form-label名前を入力してください
                    ">:</label>
                <input type="text" id="name" name="name" class="form-control"
                    required>
```

```
            </div>
            <button type="submit" class="btn btn-primary w確認-100"></button>
        </form>
        <div class="text-center mt-4">
            <a href="/list" class="btn btn-secondary追加済みの名前を見る"></a>
        </div>
    </div>
</body>
</html>
'''

@app.route('/confirm', methods=['POST'])
def confirm():
    name = request.form.get('name')
    if not name:
        return '''
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <titleエラー></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center text-dangerエラー"></h1>
        <p class="text-center名前が入力されていません。戻って再試行してください。
            "></p>
        <div class="text-center mt-4">
            <a href="/" class="btn btn-danger戻る"></a>
        </div>
    </div>
</body>
</html>
'''
    return f'''
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <title確認></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center確認"></h1>
        <p class="text-center本当に「"><b>{name}</b」を追加しますか？></p>
        <div class="text-center mt-4">
            <form method="POST" action="/add" class="d-inline">
                <input type="hidden" name="name" value="{name}">
                <button type="submit" class="btn btn-success追加"></button>
            </form>
            <a href="/" class="btn btn-secondaryキャンセル"></a>
        </div>
```

```
270     </div>
271 </body>
272 </html>
273 '''
274
275 @app.route('/add', methods=['POST'])
276 def add_name():
277     name = request.form.get('name')
278     if not name:
279         return '''
280 <!DOCTYPE html>
281 <html lang="ja">
282 <head>
283     <meta charset="UTF-8">
284     <meta name="viewport" content="width=device-width, initial-scale=1.0">
285     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
            bootstrap.min.css" rel="stylesheet">
286     <titleエラー></title>
287 </head>
288 <body>
289     <div class="container mt-5">
290         <h1 class="text-center text-dangerエラー"></h1>
291         <p class="text-center名前が入力されていません。戻って再試行してください。
                "></p>
292         <div class="text-center mt-4">
293             <a href="/" class="btn btn-danger戻る"></a>
294         </div>
295     </div>
296 </body>
297 </html>
298 '''
299
300     # 形式と形式に追加 PDPSimple
301     formatted_name_pdp = format_name_pdp(name)
302     result_pdp = add_entry_to_htkdic(formatted_name_pdp, name, pdp_dic_file)
303
304     formatted_name_simple = format_name_simple(name)
305     result_simple = add_entry_to_htkdic(formatted_name_simple, name,
            simple_dic_file)
306
307     # トリガーファイルの重複チェック後に追記
308     existing_words = set(load_trigger_words())
309     stop_julius()
310     time.sleep(0.5)
311     stop_python()
312     start_julius()
313     time.sleep(0.5)
314     start_python()
315
316     if name not in existing_words:
317         with open(trigger_file, "a", encoding="utf-8") as f:
318             f.write(name + "\n")
319         trigger_result = f名前" \"{name}\" をトリガーファイルに追加しました。"
320     else:
321         trigger_result = f名前" \"{name}\" は既にトリガーファイルに存在します。"
322
323     return f'''
324 <!DOCTYPE html>
```

```
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <title結果></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center結果"></h1>
        <div class="mt-4">
            <p class="alert alert-success">形式 PDP: {result_pdp}</p>
            <p class="alert alert-success">形式 Simple: {result_simple}</p>
            <p class="alert alert-info">{trigger_result}</p>
        </div>
        <div class="text-center mt-4">
            <a href="/" class="btn btn-primary戻る"></a>
        </div>
    </div>
</body>
</html>
’’’

#### ‘/list‘ の部分
@app.route(’/list’)
def list_names():
    trigger_words = load_trigger_words()
    if not trigger_words:
        return ’’’
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <title名前リスト></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center名前リスト"></h1>
        <p class="text-centerまだ名前が追加されていません。"></p>
        <div class="text-center mt-4">
            <a href="/" class="btn btn-secondary戻る"></a>
        </div>
    </div>
</body>
</html>
’’’

    name_options = "".join(f’<option value="{name}">{name}</option>’ for name
         in trigger_words)
    return f’’’
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <title名前リスト></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center名前リスト"></h1>
        <form method="POST" action="/delete" class="mt-4">
            <div class="mb-3">
                <label for="name" class="form-label削除する名前を選択してくださ
                    い">:</label>
                <select id="name" name="name" class="form-select">
                    <option value="">-- 選択してください--</option>
                    {name_options}
                </select>
            </div>
            <div class="mb-3">
                <label for="custom_name" class="form-labelまたは名前を直接入力し
                    てください">:</label>
                <input type="text" id="custom_name" name="custom_name" class="
                    form-control">
            </div>
            <button type="submit" class="btn btn-danger w削除-100"></button>
        </form>
        <div class="text-center mt-4">
            <a href="/" class="btn btn-secondary戻る"></a>
        </div>
    </div>
</body>
</html>
’’’

@app.route(’/delete’, methods=[’POST’])
def delete_name():
    selected_name = request.form.get(’name’)
    custom_name = request.form.get(’custom_name’)

    # 入力または選択された名前を取得
    name_to_delete = custom_name if custom_name.strip() else selected_name

    if not name_to_delete:
        return ’’’
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
    <titleエラー></title>
</head>
<body>
    <div class="container mt-5">
        <h1 class="text-center text-dangerエラー"></h1>
        <p class="text-center削除する名前が指定されていません。"></p>
        <div class="text-center mt-4">
            <a href="/list" class="btn btn-danger戻る"></a>
```

```
433         </div>
434     </div>
435 </body>
436 </html>
437 '''
438 
439     # ファイルから名前を削除
440     delete_entry(name_to_delete, trigger_file)
441     delete_entry(name_to_delete, pdp_dic_file)
442     delete_entry(name_to_delete, simple_dic_file)
443 
444     stop_julius()
445     time.sleep(0.5)
446     stop_python()
447     start_julius()
448     time.sleep(1)
449     start_python()
450 
451     return f'''
452 <!DOCTYPE html>
453 <html lang="ja">
454 <head>
455     <meta charset="UTF-8">
456     <meta name="viewport" content="width=device-width, initial-scale=1.0">
457     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
        bootstrap.min.css" rel="stylesheet">
458     <title削除成功></title>
459 </head>
460 <body>
461     <div class="container mt-5">
462         <h1 class="text-center削除成功"></h1>
463         <p class="alert alert-success text-center名前
            "> "<b>{name_to_delete}</b>" を削除しました。</p>
464         <div class="text-center mt-4">
465             <a href="/list" class="btn btn-secondary戻る"></a>
466         </div>
467     </div>
468 </body>
469 </html>
470 '''
471 
472 
473 if __name__ == '__main__':
474     try:
475         # Wi-のアドレスを取得 FiIP
476         start_julius()
477         time.sleep(3)
478         start_python()
479         local_ip = get_wifi_ip('wlan0') # Raspberry の PiWi-インターフェースは通
            常 Fi 'wlan0'
480         print(f"Starting server at http://{local_ip}:5000") # 確認用のメッセー
            ジ IP
481         app.run(host=local_ip, port=5000)
482     except RuntimeError as e:
483         print(fエラー": {e}")
```

Program 6.2: NameRecoVibe.py

```
# このコードは以下のサイトを参考にして作成した。
# 参考元: https://nekonogorogoro.com/julius_setup/

import socket
import time
import re
import gpiozero

#サーバーの接続情報 Julius
host = '127.0.0.1' ローカルホスト#
port = 10500 #のポート番号 Juliusソケット作成

#
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#のサーバーに接続 Julius
client.connect((host, port))

#の認識結果を抽出するための正規表現 Julius
extracted_word = re.compile('WORD="([^"]+)"')認識結果を格納する変数

#
data = ""

#ピンを制御するための設定 GPIO18
pin = gpiozero.DigitalOutputDevice(pin=18)トリガーワードが保存されているファイルのパス

#
trigger_file = "/home/dlite3/julius/trigger_words.txt"トリガーワードをファイルから読み込む

#
def load_trigger_words():
    try:
        with open(trigger_file, "r", encoding="utf-8") as f:
            return [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print(fトリガーファイル" {trigger_file} が見つかりません。")
        return []

try:
    while True:トリガーファイルを読み込み、トリガーワードに設定
        #
        trigger_words = load_trigger_words()

        #からデータを受信 Julius
        while (data.find("</RECOGOUT>\n.") == -1):
            data += str(client.recv(1024).decode('shift_jis'))認識結果

        #
        recog_text = "".join(extracted_word.findall(data))
        print認識結果 (": " + recog_text)認識結果にトリガーワードが含まれているかチェック

        #
        for trigger in trigger_words:
            if trigger in recog_text:
                print(f反応：特定の言葉" '{trigger}' が認識されました！")
```

```
56                  pin.on() 振動モーターを#ON
57                  time.sleep(1)
58                  pin.off() 振動モーターを#OFF
59          data = "" 格納されたデータをリセット#
```

（※文責：櫻田空大）