卒業論文

職場環境改善を支援する
小型ウェアラブルICT機器の開発による
短期ストレスへのコーピングと中長期ストレスとの関連

Development of a Small Wearable ICT Device to Support Workplace Environment Improvement: Coping with Short-Term Stress and its Relationship with Medium and Long-Term Stress

富山県立大学 工学部 電子・情報工学科
1815044 瀧田 孔明
指導教員 António Oliveira Nzinga René 講師
提出年月: 令和4年 2月

目 次

図一覧

表一覧

記号一覧

以下に本論文において用いられる用語と記号の対応表を示す.

用語	記号
n 次元上の点	P, Q
n 次元上の点をベクトル表記したもの	$\vec{x}, \vec{y}$
$\vec{x}, \vec{y}$ の k 番目の要素	x_k, y_k
解析対象となる信号	$x(t)$
基本ウェーブレット	ψ
スケールパラメータ	a
シフトパラメータ	b
定数	ω_0
抽出周波数	f
定数値	λ

第1章

はじめに

§ 1.1　本研究の背景

アンビエントコンピューティングは人の手に関わらず，機械が学習することで自動的にシステムを動かすことができる仕組みである [1]．現在よく認知されているユビキタス社会は人間側からアクションを起こし情報にアクセスする環境を表す概念である．これに対しアンビエント社会は，機械側が人間を感知し，機械側から自律的に働きかけるような社会のことを指す．

世間では IoT という概念で浸透しているが，中にはまだ人間の動作が含まれるものもあり，アンビエントコンピューティングが普及すれば機械の自動化や，作業の効率化が期待される．アンビエントコンピューティングの実用化が増えれば，機械の自動認識や人間の負担を軽減できるのでより豊かな社会を作ることができる．

現在，情報通信技術が発展しあらゆる作業が効率化され，コンピュータを用いて作業することが増えてきている．これに伴い，世界でも Society 5.0 といった AI やロボットを用いて様々な社会問題を解決する豊かな生活を目指す取り組みがなされている．現代社会では，誰もがインターネットを利用しており，役所の手続きをネット申請で行うなどインターネットが常にどこかに使用されている．今後はさらにインターネットが普及していき，ほとんどのことがインターネットを通して行うことになると考えられる．

さらに近年では，コロナウイルスの蔓延により飲食店のタッチパネル注文，仕事でのオンライン会議，自宅勤務，学校ではオンライン授業などインターネットを使ったシステムが急速に増加した．これらのシステムはウイルス感染の危険性がなくなれば元の形態に戻るというのもあるが，一方でオンラインシステムを使った方がより効率が上がるということで今後も継続していくものも一部あると考えられる．このようにオンラインシステムは今後増えると予想され，社会環境はますます変化していくことが考えられる．

しかしながら，オンライン事業が増えると同時に長時間のデスクワークによる目の疲れや精神的・身体的な疲労が問題となってくる．このまま問題が増え続けてしまうと深刻な問題になる．オンライン授業では，画面の注視や受講環境を整えることや課題の提出に戸惑うなど解決しなければならない問題がたくさんある．

また，長時間同じ体勢を続けると起こる可能性のあるエコノミークラス症候群は，デスクワークなどでずっと座り続けるような状態では起こりやすいと考えられる．座って作業をすることが多い現代社会では特に気を付けるべきである [2]．さらに，デスクワークに限らず同じ行動の長時間の継続（運転など）は精神的・身体的負担がかかることが研究されている [3]．この問題を解決するために，心拍センサを用いたりしてストレス値を測定して

コーピング指示によりストレスを緩和させるような研究が存在する [4].

このようにウェアラブル装置でストレス計測を行うことは，ストレスの原因を明確に知ることができストレスが溜まってしまう前に迅速にストレス対処を行うことができ，人間が特に操作することはないという利点がある．ウェアラブル装置を開発してストレス計測を行う方法はたくさん存在する．既存の研究では，長時間インターネットの使用を行うネット依存と交感神経による興奮状態には関係性があることが証明された [5].

ウェアラブル装置のセンサとコーピング手法を使用することにより，正しいストレス軽減を行うことが容易になりストレスに対してうまく立ち回ることができるようになると思われる．今後センサでストレス測定を行うことが一般的になれば，ストレスの対処方法が明確になりより豊かな社会を目指しやすくなると考えられる．

§ 1.2　本研究の目的

本研究では，何人かの人に様々なセンサが付属した小型化ウェアラブル装置を身体の一部に取り付けてもらい，短期のストレスを測定してコーピング指示を出すこととストレスチェックシートより中長期のストレス測定を行って双方のストレス負荷の順位が一致することを目的とする．ウェアラブル装置のような機械でストレス測定することとストレスチェックシートのような人間自身の判断でストレスチェックを行い，双方が一致すれば本研究で作成したウェアラブル装置は正しいストレス測定が行えていると考えられる．そこで本研究ではコーピングによるストレス軽減の方法の提示とストレッサーの発見を明確にするようなシステムを開発する．

本研究では，センサ，カメラ，マイクを用いて逐一ライフログを蓄積させていく．この記録するライフログを参考に行動識別を行う．行動識別を行うことにより，ストレッサーの発見がより容易になる．コーピング指示を行うことにより，ストレス緩和をしやすくなり日常生活においてストレスがかかってしまう状況を減らすことができる．またストレスチェックシートから人間の主観的な判断からもストレスチェックを行えるため，ウェアラブル装置の測定と合わせてより具体的なストレス測定ができることがうかがえる．

本研究の具体的なストレス測定方法として，短期ストレスの測定は小型化ウェアラブル装置を使用してその中の心拍センサを用いて一般的なストレス指標である LF/HF 値を求める．この LF/HF 値をもとにして，コーピング指示が計測中に何回出たかで短期ストレスを評価する．また中長期ストレスは，ストレスチェックシートのそれぞれの設問を 4 択として点数配分を決めてその合計点数が高い人ほどストレスがかかっているという評価にする．これら 2 つの結果をそれぞれ順位付けし，本研究のウェアラブル装置に有効性があることを示す．

LF/HF 値やマイクでの行動識別を記録したら，コーピング指示を行うようにする．今回のコーピング指示は視覚と聴覚の 2 点を使って行う．視覚表示では，単的な理解しやすい言葉と図を用いて自身のストレス状態をよりわかりやすいようにする．また聴覚での指示は小型化ウェアラブル装置にイヤホンを付けることとし，コーピング指示が出された時とストレス値が上昇してきた際の注意喚起の 2 点で自動的に音声出力がされるようにする．

§ 1.3　本論文の概要

本論文は次のように構成される．

第 1 章 本研究の概要と目的について説明した．概要は，インターネットの普及による利便性やオンラインでの作業増加，またこれらが普及することでの問題点について述べた．目的は，短期ストレスと中長期ストレスの比較の実用性，小型化したウェアラブル装置の主な役割，コーピング指示での視覚と聴覚のそれぞれの指示内容の説明について述べた．

第 2 章 行動識別をする上でのライフログのデータ収集やその概要についての先行研究を説明する．また，コーピングの理論について説明し，ストレス値の算出方法について説明する．

第 3 章 現在ストレスチェックの一般的な方法について述べ，ストレスチェックシートの質問内容，評価方法について説明する．その上で，本研究で行うようなウェアラブル装置を用いたストレス測定においての利便性を述べる．

第 4 章 本研究での行動識別，ストレス計測，コーピング内容の決定の仕方などシステムの提案手法を述べる．また，短期ストレスと中長期ストレス測定のそれぞれのアルゴリズムについても説明する．

第 5 章 提案手法に基づいて，短期ストレスと中長期ストレスのどちらともを計測し，双方のストレス負荷の順位が一致することで本研究のウェアラブル装置のストレス測定に有効性があるかを述べる．

第 6 章 本研究で行ったことと今後の課題を述べる．

第2章

行動識別を用いたストレスコーピング

§ 2.1　センサを用いたライフログ収集

人間のライフログを記録してそのデータを用いて分析するというものはたくさん存在する．ライフログの内容としては，心拍数を記録する，血圧を測定してくれる，血中酸素濃度を記録するなど内容は様々なものがある．これらのものを記録することによって自分自身の健康状態や日常の生活習慣をより良くすることを目的とするものが多い．

このようなシステム例としてスマートウォッチというものがある．このスマートウォッチを用いて 1 日あたりの活動量としての歩数，消費カロリー，睡眠時間，睡眠サイクルなどを記録してライフログを活用している研究が存在する [6]．このように，装着しても邪魔にならないようなウェアラブル装置というものは様々な種類が存在し，またライフログデータを収集して生活習慣や日常生活を正しくしようと働きかけるシステムは多数存在する．

しかし現在のライフログを収集するようなシステムではユーザが操作しなければならない部分やライフログを収集してからの改善方法があまりうまく行われていない場合が多く存在する．この問題の解決方法は，機械の自動分析，収集した後の具体的な改善策の提示をすることである．

先行研究では，様々なセンサを取り付けて多種多様な測定が行える LLC 社の「Arduino Uno」と Raspberry Pi Foundation によって開発された「Raspberry Pi 3 (Model B)」の 2 つのマイコンを使用し，Arduino に体温や心拍数を測定する生体センサと照度や加速度など周囲の環境を測定する環境センサを取り付け，身体の上半身にこれらのウェアラブル装置を装着してストレス測定を行う研究が存在する [7]．本研究では，先行研究のウェアラブル装置の小型化を行うため「Arduino Uno」から「Arduino nano」へ，また「Raspberry Pi 3 (Model B)」から「Raspberry Pi Zero WH」へと変更し，ストレス測定に不要だと思われるセンサをいくつか取り除いたものを使用した．

Raspberry Pi Zero WH

Raspberry Pi Zero WH とは，Raspberry Pi 3 (Model B) と比較するとかなり小さくなったものであり，低価格，省電力のモデルである．主にプログラミング環境や電子工作で使用されており，この小さいマイコンボードでもカメラを取り付けたり，microHDMI ケーブルを用いてテレビ画面の表示などができる．無線 LAN や GPIO ピンも搭載されているため，データの送受信を行えたりまた GPIO ピンに測定したいセンサを取り付けたりして測定を行うことも可能である．

Arduino nano

Arduino nanoとは，Arduino Unoと比べてさらにコンパクトになり，ブレッドボードに直接差し込んで利用できるものとなっている．Arduino UnoとI2C通信やシリアル通信など様々な通信方法を使用することができ，また容量の面でもオーバーしてしまうことがなかったので今回は小型化をするためにArduino nanoを使用した．

生体情報として用いるセンサは，体温， 心拍，ガルバニック皮膚反応の3つのセンサである．これらのセンサは人間の体調を調べるに当たって重要なセンサとなる．また環境情報として用いるセンサは温湿度気圧，照度，加速度，カメラ，マイク，イヤホンの6つのセンサである．先行研究で使用していた人感，GPSセンサについては，行動識別において直接的に必要ではなかったため取り除いた．

上記の生体センサと環境センサである9つのセンサを使用することで，行動識別を明確に識別することができ，またコーピング指示を出す際にセンサの値と行動識別からより適切なコーピング指示を出すことが可能となる．新たにウェアラブル装置にイヤホンを付属させることによってコーピング指示のパソコン画面などの視覚表示を見落としてしまった場合でもコーピング指示を伝えることができるようになった．

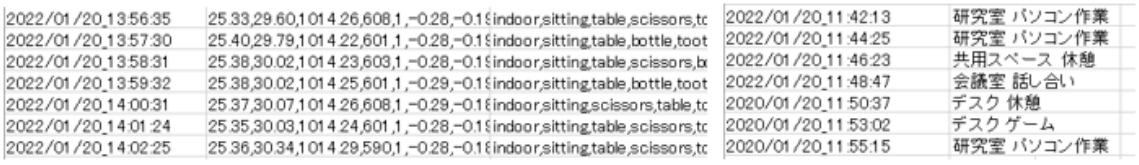
取得するデータ一覧

・環境センサ
温度，湿度，気圧，照度，加速度（3軸），角速度（3軸），地磁気（3軸），カメラ，Respeaker 2-Mics Pi HAT(音声入力用マイク)
・生体センサ
体温，心拍，ガルバニック皮膚反応

上記の8つセンサとArduino nanoとRaspberry Pi Zero WHを接続して計測を行う．Arduino nanoで温度，湿度，気圧，照度，体温，心拍，ガルバニック皮膚反応，加速度(3軸)，角速度(3軸)，地磁気(3軸)のセンサといった計6つのセンサを接続し16種類のデータを収集させ，Raspberry Pi Zero WHにはカメラ，マイクの計2つのセンサを接続し2種類のデータを収集させるようなウェアラブル装置を開発する．

本研究のライフログのデータ収集の具体的な手法として，2つのマイコンボードをケーブルで接続し，シリアル通信でデータ収集を行うものとする．Arduino nanoに取り付けたセンサのデータをまずはRaspberry Pi Zero WHにシリアル通信で送信し，その後Raspberry Pi Zero WHで収集しているセンサデータと合わせてWi-Fi環境を用いて研究室のクラウドサーバにセンサデータを送信する．この時にRaspberry Pi Zero WHでは，クラウドサーバにセンサデータを送信することとArduino nanoからのセンサデータを取得するプログラムを実行しているが，これはPCを使用してVNC viewerというアプリケーションを使って遠隔操作で実行している．

サーバに送信されたセンサデータは，サーバ上に作成したcsvファイルに毎回記録されるようなシステムとなっている．このcsvファイルの中身は図2.1に示すように生体・環境センサのデータ合わせて19のデータとサーバに送られてきた時間と合わせて記録するものとなっている．音声データについては音声入力した場合にサーバにデータを送信する仕組



2022/01/20_13:56:35	25.33,29.60,1014.26,608,1,-0.28,-0.1§indoor,sitting,table,scissors,tc	2022/01/20_11:42:13	研究室 パソコン作業
2022/01/20_13:57:30	25.40,29.79,1014.22,601,1,-0.28,-0.1§indoor,sitting,table,bottle,toot	2022/01/20_11:44:25	研究室 パソコン作業
2022/01/20_13:58:31	25.38,30.02,1014.23,603,1,-0.28,-0.1§indoor,sitting,table,scissors,b	2022/01/20_11:46:23	共用スペース 休憩
2022/01/20_13:59:32	25.38,30.02,1014.25,601,1,-0.29,-0.1§indoor,sitting,table,bottle,toot	2022/01/20_11:48:47	会議室 話し合い
2022/01/20_14:00:31	25.37,30.07,1014.26,608,1,-0.29,-0.1§indoor,sitting,scissors,table,tc	2020/01/20_11:50:37	デスク 休憩
2022/01/20_14:01:24	25.35,30.03,1014.24,601,1,-0.28,-0.1§indoor,sitting,table,scissors,tc	2020/01/20_11:53:02	デスク ゲーム
2022/01/20_14:02:25	25.36,30.34,1014.29,590,1,-0.28,-0.1§indoor,sitting,table,scissors,tc	2020/01/20_11:55:15	研究室 パソコン作業

図 2.1: センサデータシート (左) とマイクからのテキストデータシート (右)

みとなっているので，次の生体・環境センサデータを記録する際に同じ行に記録している．

§ 2.2　ストレスコーピング理論

コーピングとは，ストレスを対処するための行動のことを指す．コーピング手法は，大きく分けて 3 つありストレスに対して自身の努力や周囲の協力によって解決や対策に取り組む問題焦点型コーピングとストレスそのものに焦点を当てるのではなく，問題を捉える自分自身の感情に焦点を当てる情動焦点型コーピングとストレスになっているものから離れることや，ストレスの緩和などで対策を取るストレス解消型コーピングの 3 つとなる [8]．

コーピングの重要性として，ストレスがかかっている時にコーピングを実行することでストレス緩和につながるというストレスとコーピングの相互の関係性が存在することは先行研究にて証明されている [9]．本研究では，3 つのコーピング手法の内の問題焦点型コーピングという手法を用いて，センサの値やマイク，カメラでの行動識別より適切なコーピング指示を決定してストレッサーとなっている問題を直接解決できるようなシステムを開発する．

また，先行研究において心拍取得用のウェアラブルデバイスを装着してもらい，VDT 機器を操作している最中に LF/HF 値よりストレス値を算出し，LF/HF 値が一定より高くなったら PC から警告音を流し，ストレス緩和のためのストレッチを促すという研究がある [4]．この研究を参考にコーピング指示において視覚だけでのみコーピング指示を確認できるようにするだけでなく，ウェアラブル装置にイヤホンを装着できるようにし，作業をしている最中でもコーピング指示の見落としがないような適格にコーピング指示を出せるようにする．

コーピングを行う際にマイクの音声入力とカメラ画像を用いた行動識別を利用するが，この行動識別の手法を階層的クラスター分析とする．階層的クラスター分析とは，あるデータ群の中から最も似ているデータ同士を順にまとめていき，徐々にクラスターの数を少なくしていく手法である．この似ているデータ同士を結合していく際に本研究では「樹形図 (デンドログラム)」を作成し，結合しているデータを図でも確認できるようにする．

また，階層クラスター分析での分類するデータ間の類似度としてユークリッド距離を使用する．ユークリッド距離とは，2 点をつないだ直線距離を表したものであり，下記の式 2.1 によって計算することができる．また，クラスターを結合する方法としてはウォードを用いる．このウォード法とは，全てのクラスターの中で最も距離が近い 2 つのクラスターを 1 つのクラスターに合体させていき，最終的に 1 つのクラスターにまとめるといった手法である．

$$d(P, Q) = |\overrightarrow{x} - \overrightarrow{y}| = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{2.1}$$

デンドログラムの作成については，カメラから取得した静止画像をテキスト変換し，このテキストを word2vec を用いてさらに数値ベクトルに変換し他のセンサデータとともに記録する．ここで，デンドログラムにおける各クラスターのラベルはマイクの音声入力によって得られたテキストを使用する．

また，最新のクラスターのラベルに関しては “最新” のラベルになるようにした．マイクの音声入力については，一定な入力ではないので他のセンサデータを送信するのとはまた別にクラウドサーバに送信できるようなプログラムを作成する．この音声入力プログラムでは，周りのノイズにより間違ったマイクラベルを取得しないように，ノイズ除去をしてくれる文章とホットワードを取り入れることとする．

まず，マイクに向かってホットワードを言い，正しく入力できたら自身の状態と場所をマイクに入力する．この 2 つが正しく入力されたら，再度またホットワードを入力させるようにしこれを繰り返すこととする．ホットワード，場所と状態の入力全てがラベル化できた時にマイク用の Respeaker 2-Mics Pi HAT の LED が光るようにし，どこでマイク入力をしても判断ができるようにする．
このマイクのラベリングを使用することで現在の行動がどのクラスターに分類されるのかが分かり，行動識別が可能となる．

本研究での画像認識と音声認識ではそれぞれ Computer Vision API と Google Speech API を使用する．Computer Vision API とは，Microsoft 社が提供している API の 1 つであり，リアルタイムの画像分析，テキスト抽出の自動化などを実現することができる．また Google Speech API とは，Google が開発した API であり，音声入力をしたものをテキスト変換することができる．

静止画像をテキスト化し，さらにベクトル化をする際に wod2vec を使用する．分散表現のモデルは，学習済みモデルを使用しており，手法として Skip-Gram Model を用いて次元数は 300 で表される [10]．また，word2vec を使用するに当たっての今回の手法は，Skip-Gram Model とする．300 次元となってしまうと，クラスター分析を行う際にセンサデータにノイズが発生してしまうため，規準の単語をいくつか設定し，類似度によって次元数が減少された単語をクラスター分析する．規準の単語は，日常的に関わりがある furniture，animal，plant，behave，food，appliance(家具，動物，植物，行動，食品，家電) の 6 つとした．静止画像を Computer Vision API を用いてテキスト変換した後，各テキストと 6 つの規準単語ごとに類似度を足し合わせ，平均を求める．この手法を行うことにより，カメラの静止画像は 6 次元で表すことが可能となる．

実際にパソコン作業とゲームをしている時の静止画像を word2vec によって 6 次元の数値ベクトルに変換して比較してみたものを表 2.1 に示す．表 2.1 の内容は，2 つの静止画像をテキスト変換したものと 6 つの規準単語のそれぞれの類似度の平均を表している．パソコンで作業をしている時には，computer，monitor，keybord といったパソコン関係のテキストが多く保存されていることやまた，ゲームをしている時には appliance の値が高くなっているのが分かる．

表 2.1: 行動の違いによる数値の比較

基準		furniture	animal	plant	behave	food	appliance
パソコン	テキスト	indoor, sitting, monitor, computer, television, screen, table,…					
	数値化後	0.2756	0.1690	0.1314	0.1574	0.1728	0.2349
ゲーム	テキスト	indoor, light, sitting, small, hanging, room, table,…					
	数値化後	0.3553	0.1959	0.1747	0.1548	0.2378	0.3128

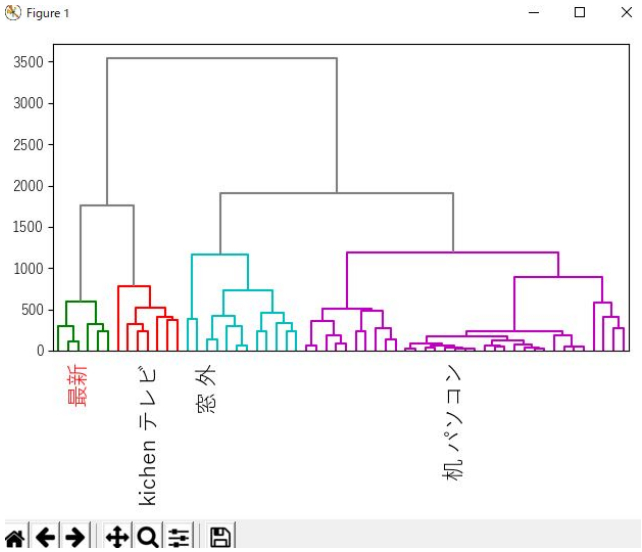


図 2.2: デンドログラム

結果として，静止画像を 6 次元に変換してみると静止画像に対しての特徴が際立ったため，行動識別を行うことができる．この 6 次元のテキスト変換データと生体・環境センサデータとマイクのラベルを 1 つに結合したライフログのデータを作成する．このライフログのデータは，研究室のクラウドサーバに蓄積され，このライフログデータを基にして適切な問題焦点型コーピングを行う．

図 2.2 では，上記のライフログのデータを基にしてクラスター分析を行った際の結果をデンドログラムで表した結果を示す．今回はユークリッド距離を 2000 に設定した．理由としては，同じ室内にいても違う行動を識別できる距離だからである．

デンドログラムには，マイクに入力したラベルがそのままクラスターの名称となるようにし，どのような行動識別ができたのかを可視化して見やすくして表示する．

§ 2.3　ウェーブレット変換によるストレス値算出

本研究でのストレスの捉え方として，交感神経と副交感神経の 2 つを参考にして交感神経が優位な場合，血管が収縮して血圧が上昇し，身体が活動的になっている状態，副交感神経が優位な場合は，血管が緩んで血圧が低下し，身体がリラックスしている状態を表し

ているものとする．

ストレス計測を行う方法として，現在の社会ではストレスチェックシートを用いたストレス測定が一般的である [11]．しかし，このストレスチェックシートのみでのストレス測定は年に 1 回行うなど回数が少ないことからいつどのようなストレスに悩まされたのかという判別が難しかったり，また会社では繁忙期とそうでない時期とでは人間の主観によるストレスチェックシートの回答のみでは正しいストレス測定が行えていない場合が生じてくる．よって，ストレスチェックシートに加えてセンサでの測定を行ったらより精度の良いストレス測定が行える．

また，センサを用いたストレス測定を行っている既存の研究もいくつか存在する．その中の 1 つにストレスマーカーを使用して唾液から検査を行う研究がある [12]．唾液アミラーゼ活性を測定することで，ストレス度を評価しており，アミラーゼは炭水化物の消化酵素であるが，ストレスを感じた時に上昇することが分かっており，唾液で測定することができる．この唾液アミラーゼ活性は交感神経の状態を反射し，身体的・精神的な刺激に対して短時間で反応する特徴がある．このような唾液や血液のような人間の身体から採取してストレス測定を行う研究も存在する．

他のストレス測定方法である皮膚温では，指先などの抹消部皮膚温を計測する手法を取る．抹消部皮膚温はストレスを感じるような心理的要因によって低下することが証明されている [13]．本研究で使用しているガルバニック皮膚反応は，緊張などで発汗してしまう等で変化する皮膚上の電気抵抗を計測する仕組みとなっている．

ストレス値を算出するに当たってこの計算方法に自律神経のバランスを推定する HF と LF が用いられる [14]．まず，HF とは High Frequency の略称で高周波を表し，3 秒から 4 秒程度の周期を持つ呼吸を信号源とする変動波である．または，その周波数領域のパワースペクトルの合計量を表す．一方で LF とは，Low Frequency の略称で低周波を表し，メイヤー波と呼ばれる約 10 秒周期の血圧変化を信号源とする変動波である．

また，その周波数領域のパワースペクトルの合計量を指す．交感神経と副交感神経の緊張状態のバランスにより，心拍変動への HF の変動波と LF の変動波の現れる大きさが変わってくるため，この変化を利用して心拍変動から自律神経のバランスを推定する．ここで扱う「ストレス」の定義は，交感神経と副交感神経の緊張の具合，つまりバランスであり交感神経が緊張状態であれば「ストレス状態」，逆に副交感神経が緊張状態にあれば「リラックス状態」という定義を採用している．

パワースペクトルの算出方法として，まず心拍間隔変動時系列データを作成する．心拍センサを用いて，心拍データが出力されることからこれを 60 で割ることで心拍間隔である RRI が求められる．続いて心拍間隔変動時系列データからパワースペクトル密度 (power spectral density: 以下 PSD と略す) を算出する．

本研究では，パワースペクトル密度を推定するためにウェーブレット変換という手法を用いた [15]．一般的なウェーブレット変換 $W(a,b)$ は，式 (2.2) で表される．

$$W(a,b) = |a|^{\frac{-1}{2}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-b}{a}\right) dt \tag{2.2}$$

$x(t)$ は解析対象となる信号を表し，ψ は基本ウェーブレットである．そして，a はスケールパラメータ，b はシフトパラメータである．スケールパラメータ a は抽出対象周波数に対応しており，b は解析対象時刻に対応する．ウェーブレット変換をするに当たって基本ウェー

ブレットを 1 つ用いる必要があり，本研究ではガボール関数を用いた．ガボール関数の式を式 (2.3) に表す．

$$g(t) = \exp\left(-\left(\frac{t-b}{a}\right)^2\right)\exp\left(-j\omega_0\frac{t-b}{a}\right) \tag{2.3}$$

ここでの a, b も式 (2.2) と同様にそれぞれスケールパラメータとシフトパラメータである．また ω_0 は定数である．ウェーブレット変換は，スケールパラメータとシフトパラメータの関数であるが，生体信号における周波数成分のパワーあるいは振幅の時間-周波数解析においては，スケールパラメータ a は抽出周波数 f を用いて $a = 1/f$ と表すのが一般的である．よって，ウェーブレット変換の式を周波数 f とシフトパラメータとの関数とし，ω_0 を 2π とした式を式 (2.4) に表す．

$$W(f,b) = \left|\frac{1}{f}\right|^{\frac{-1}{2}}\int_{-\infty}^{\infty} x(t)\exp\left(-\left(\frac{t-b}{1/f}\right)^2\right)\cdot\exp\left(-j2\pi\frac{t-b}{1/f}\right)\,dt \tag{2.4}$$

ここからパワースペクトルを推定するためには，最適な解析データ長である窓幅について検討する必要がある．そこで，窓幅とパワースペクトルとの関連を検討するためにガウスの広がりを決定している $1/f$ に定数値 λ を導入した．その式を式 (2.5) に示す．

$$W(f,b) = \left|\frac{1}{f}\right|^{\frac{-1}{2}}\int_{-\infty}^{\infty} x(t)\exp\left(-\left(\frac{t-b}{\lambda/f}\right)^2\right)\cdot\exp\left(-j2\pi\frac{t-b}{1/f}\right)\,dt \tag{2.5}$$

これらの式を基にして，各心拍の PSD から HF 成分と LF 成分の大きさを求めることができる．本研究での LF と HF の規準値は一般的な領域とし，LF の領域を 0.05 Hz - 0.15Hz, HF の領域を 0.15 Hz - 0.40 Hz としている (図 2.3 参照)．これらの HF 成分と LF 成分をそれぞれ合計したものを HF，LF とする．

この 2 つの比 LF/HF はストレス指標である．リラックスしている状態では HF 成分が増加するが，ストレス負荷がかかっている時には LF 成分の方が増加する．よってリラックス状態の時には LF/HF 値は小さくなり，ストレス負荷がかかっている状態では LF/HF 値は大きくなる．

LF/HF の評価方法は，疲労科学研究所の規準値がよく使われるため，本研究でもこの LF/HF 値の評価方法を参考にし，ストレス段階を良好，注意，要注意の 3 つに分類して評価していくものとする (表 2.2 参照)．

図 2.4 はウェーブレット変換をした際のパワースペクトルを表したものである．多少ばらつきがあるのは，ウェーブレット変換をする際に 1024 の心拍時系列データを溜めたことによるものだと考えられる．また，0.4 以上のところに多少波形が出てしまっているが，この部分の波形はプログラム上で範囲を決めて積分を行っているためストレス値の算出の際には使用していない．図 2.5 は，ウェーブレット変換によるパワースペクトルのカラー画像を表したものである．

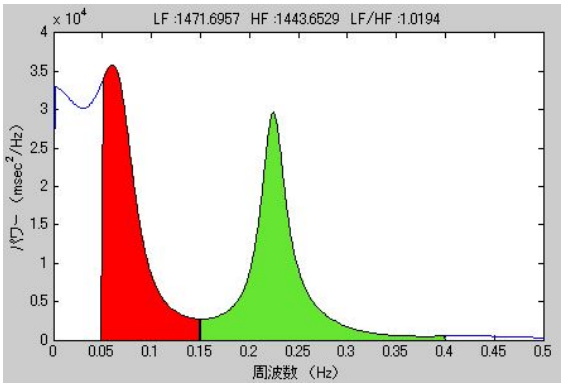


図 2.3: PSD と LF と HF の大きさの比

表 2.2: ストレス評価

LF/HF	評価段階
0〜2.0	良好
2.0〜5.0	注意
5.0 以上	要注意

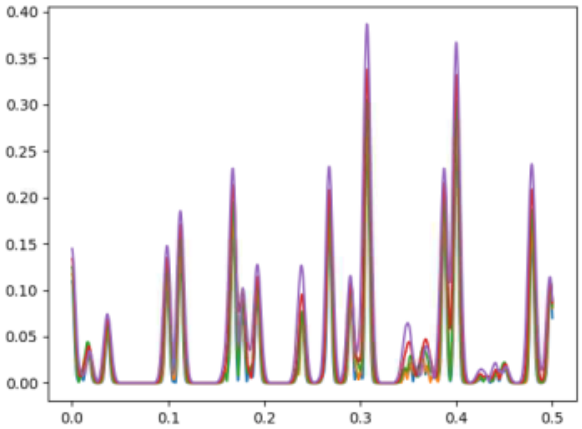


図 2.4: パワースペクトルの推定

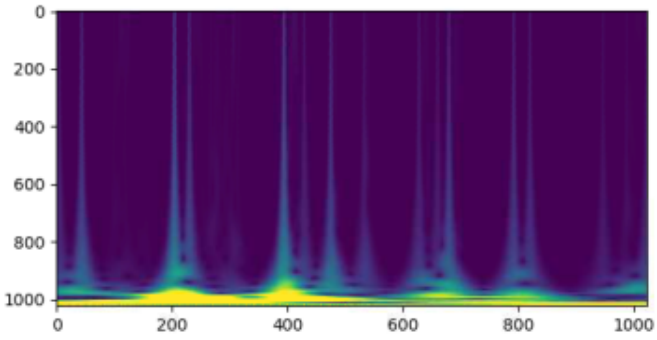


図 2.5: パワースペクトルカラー画像

第3章

アンビエントインテリジェンスとストレスコーピング

§ 3.1　健康・衛生管理のためのストレスチェック

ストレスチェックは，現在職場などにおいて実施されている．その内容としては，年に1度その年のことを振り返って自分自身で評価するというものが多い．また，2015年12月1日より「労働安全衛生法」が改正され，労働者が50人以上いる事業場では毎年1回,「ストレスチェック」を全ての働く人に対して実施することが義務づけられた [16]．さらにこのストレスチェックをしたうえで高ストレス者と判断された人と長時間労働者に対しては，医師からの面接指導を受けるように促すことになっている．このようにストレスチェックは，徐々に大切なものとなってきている．

また，自身についてのストレスチェックを行うような手法はもちろん存在するが，自分だけでなく身近な人をターゲットとしたストレスチェックも存在する [17]．ストレスを認識できる心の変化や行動として仕事をしている人なら不安な気持ちになる，落ち着かない気分が続く，以前と比べて怒りっぽくなるなど様々な心境の変化が挙げられる．また，女性の方の場合なら動悸，呼吸が苦しくなる，手足が震えるなどの症状を訴える方が多い．逆に男性では，眠れない状態が続き，寝る前に「寝るためのお酒」を飲んでしまいそのうちにアルコールに依存するようになってしまう方などがいる．このような症状が現れる背景としては，職場の人間関係や仕事の内容，仕事の忙しさや重大な仕事を抱えたことによる緊張感など人によって様々な問題が存在している．

職場や家庭でも，その人を以前と比較してみて明らかにイライラしていたり，攻撃的になったりしている場合は「ストレスが溜まっている状態」という可能性が高いと思われる．家庭環境で最近ストレスがかかってしまっている問題としては，介護が多く挙げられる．高齢の両親が病気になってしまい，介護を何年間も行うことは肉体的，精神的な負担は大きいものである．高齢化社会が進む現在，このようなケースは今後も増え続けていくと思われるためストレス軽減をする対策方法などが必要であることが考えられる．

また，職場でストレスがかかっていると思われる人がいるなら可能であれば仕事を減らしたり，仕事に余裕を持たせるような配慮が必要だと思われる．ひどくなってしまい精神が病んでしまうと仕事もできなくなるため，悪くなる前に何かしらの手を打つことが大切である．仕事ができる人ほど几帳面で完璧主義の傾向があり，そのような方はうつになりやすいと言われている．もし身近でこのような人がいるなら「大変そうですね」「なにかあったんですか？」など声をかけたりすることも良いと考えられている．受け止め方はその人によって様々ではあるが，周りの人が声をかけることによって自分自身を振り返るきっかけの一つになったりする場合がある．

図 3.1: 従来照明 (左) とタスクアンビエント照明 (右)

図 3.2: amor HP+

一方で，Apple Watch やスマートウォッチといったウェアラブル装置から脈拍測定や血中酸素濃度を測定するようなシステムも普及しつつある．これからのストレスチェックは，ストレスチェックシートに加えて機械が自動的に身体計測を行ってくれるようなものが次第に多くなってくると考えられる．

機械がセンサなどを用いて自動的に人間に何かを働きかけることをアンビエントコンピューティングという．この技術が普及しているアンビエント社会では，人間の動作なしで快適な環境，安心安全な環境を保持したりする社会である．この起源は 1998 年，アメリカの Palo Alto Ventures の Eli Zelkha と Brian Epstein がフィリップス向けのプレゼンテーション「2010 年-2020 年ごろの社会」にて定義づけされた．

アンビエント (Ambient) とは，周囲の，環境などの意味があり家電や電子機器，環境全体をコンピュータの一部とみなしたものである．アンビエントコンピューティングの先行研究では，住環境にアンビエントインテリジェンスシステムの技術を取り入れ，人間の動きを感知することで電化製品を用いて最適な環境を作るというものがある [18].

アンビエントコンピューティングの代表例は，Amazon Alexa や Google Assistant といったスマートスピーカーが有名である．これらは人間が言葉で話しかけることにより，音声認識をし人間の代わりに行動を起こしてくれるものである．本研究での音声入力マイクで使用している Respeaker 2-Mics Pi HAT は，入力した音声をテキスト変換し上記 2 つのスマートスピーカーにデータを送信するという役割も行うことができる．また図 3.1 は，アンビエントライトというものであり作業形態の多様化に対応できることと時間的な変化にきめ細かく対応できることの 2 つの基本的な考え方を持っている．作業形態の多様化や時間的変化に対応できるようタスクとアンビエントそれぞれに専用の照明手段を駆使して作業環境に適した空間を作っている．

これらは，家庭などにおいて人間を手助けできる機能を持っており，機械側から人間に働きかけていると言える．現在，アンビエントコンピューティングは世間に広まりつつあり，今後も様々な分野において活躍が期待されている．

本研究のようなストレス測定の分野では，まだまだアンビエントコンピューティングの技術を使用したウェアラブル装置の認知度は低いと考えられる．近年，若者が職場や人間関係などでストレスを感じてしまうことが多く，その結果耐えられなくなり自殺してしまうといったケースが増えてきている．これを阻止するために，ストレスに対して適切に対処していく必要がある．この解決策として，ストレスへの対処法を適切に指示してくれるようなコーピング指示があげられる．

このコーピング指示は，専門家に相談してストレス対処のアドバイスをもらうというの

が一般的ではあるが，これは常にコーピング指示を受けられるというものではない．この方法だけでは，その時に適切なコーピング指示をすることが難しいことよりうまくストレスへと対処できていない可能性が生じてくる．適切なコーピング指示を行うためには，常にストレス測定を行えるようなウェアラブル装置を使用するのが最も効果的な方法であると考え，センサ情報も交えて自身の周りの環境や状況を識別することによってより正しいストレスコーピングが行えると思われる．

ウェアラブル装置でストレスコーピングを行ってくれる「amor HP+」というリストバンド型ウェアラブルデバイスが存在する．これには，PPG センサ (光学式心拍センサ)，G センサ (加速度センサ) が内蔵されており，手首に着用することで「心拍数」,「歩数」を 24 時間リアルタイムで測定することができる．また，身体のストレス状況を表示することができてストレスレベルから運動や休憩を促す警告などをバイブレーションで通知する機能が搭載されている．

現状，身近なスマートウォッチなどを装着して身体測定を行うシステムは多数存在しているが，測定に対するフィードバックを行うシステムはまだあまり存在していないと考えられる．本研究では，コーピングにおいて上記のスマートバンドも参考にしつつバイブレーションでのコーピングは画面が見れない場合にどのようにストレス対処をするのかを確認することができないことより，イヤホンを通して音声でコーピング指示を伝え，また様々な生体・環境センサを取り入れてよりコーピングの精度を上げることとした．

§ 3.2　中長期ストレスの認知のためのストレスチェックシート

本研究の中長期ストレスの測定方法として，ストレスチェックシートを回答結果を参考にする．ストレスチェックとは，労働者のストレス状況を早期に把握し，メンタルヘルス不調のリスクを未然に防ぐために実施する検査である．本論文の 3．1 節にも記載した通り,「労働安全衛生法」という法律が改正され，2015 年 12 月から毎年 1 回，ストレスチェックを全ての労働者に対して実施することが義務付けられた．このストレスチェックを行う理由としては，労働者が自身のストレスの状態を知ることにより，ストレスがたまりすぎるのを防いだり，またストレスが高くなってしまった場合には医師の面接を受けて助言をもらったり，会社側から仕事の軽減などの措置を実施してもらうなど職場の改善につなげることで,「うつ」などのメンタルヘルス不調を未然に防止することができるからである．

またストレスチェックの流れは図 3.3 に示す．ストレスチェックと面接指導の実施状況については，毎年労働規準監督署に所定の様式で報告しなければならない決まりになっている．国が推奨している 57 項目の質問票 (職業性ストレス簡易調査票) の内容と本人に通知するストレスチェック結果のイメージは図 3.4 のようになる．この厚生労働省が推奨している「職業ストレス簡易調査票」は，1999 年に研究班が 21 企業に多職種・他業種を対象に調査を実施し，12274 名に対して回答してもらっているため信頼性，妥当性が実証されている．職場に関するストレスチェックについての法律は 3．1 節の通りとなるが，労働者が 50 人以下の事業場であるならストレスチェックを行う義務は存在しない．しかし，人数が少ないからといって複数の人間が働く環境には少なからずストレスが生じてくる．そのため，労働者の人間に関係なく，例え法律がなかったとしても定期的なストレスチェックを行い，働きやすい職場を目指して改善を重ねていくことが望ましいと言える．

ストレスチェックには，労働安全衛生規則により「仕事のストレス原因」,「心身のストレス反応」,「周囲のサポート」の 3 領域の質問事項を含むことを規定している.「仕事のストレス原因」は，仕事の作業環境や労働時間，仕事の量や人間関係などのストレスの原因に関する質問事項である.「心身のストレス反応」は，自分がどのような感情でいるのか，体に出ている症状などのストレスの反応から今の状態を調査する.「周囲のサポート」は，自分の身の回りにサポートしてくれる人や頼れる人がいるかといった項目が含まれており，今後のストレス緩和につながる可能性があるかどうかを確認する質問事項である．これらの 3 領域の質問項目を点数化し，高ストレス者を判定していく．

また，厚生労働省から出されている「職業性ストレス簡易調査票」の問題数として 23 項目版，57 項目版，80 項目版の 3 種類の問題数が一般的には多く存在している．まず 23 項目は，57 項目版の内容を簡略化したものであり最低限の 3 領域をみたしたものとなっている．57 項目版は，厚生労働省が推奨しているものでありストレスチェックに必要な 3 領域をすべて網羅しており，5 分程度で手早くストレスチェックシートに回答をし調査できるといったところがメリットとなっている．最後に 80 項目版は，個人のストレス反応だけではなく，働きがい (モチベーションなど) や上司やマネジメント，ハラスメントを測定できる．特にハラスメントに関しては，2020 年 6 月に「パワハラ防止法」が施行され，会社側が対策を講ずることが強く求められるようになった．このように職場環境の改善に重点を置くなら，80 項目版の方が良いと考えられる．今回の本研究で作成したストレスチェックシートでは，手短に回答できることストレス状況を知ることとして使用するため 57 項目版と同等な質問量でのストレスチェックシートを作成することとした．

ストレスチェックシートに回答し，ストレスチェック結果により「医師による面接指導が必要」とされた労働者は，結果が通知されてから 1ヶ月以内に面接の申し出を行い，医師に依頼して面接指導を行ってもらうことが可能である．最後にストレスチェックの実施者が，ストレスチェック結果を一定規模の集団 (部署，課，グループなど) ごとに集計・分析をしてもらい，その結果を提示する．

ストレスチェック制度は，労働者の個人情報が適切に保護され，不正利用などが絶対に行われないようにすることで労働者が安心してストレスチェックを受け，それに対して適切な対応や改善につなげられる仕組みとなっている．個人情報を守るためにプライバシーの保護，不利益取扱いの防止など様々なルールが適用されている．

ストレスチェックシートにはこのように様々な種類，ルールが適用されている．厚生労働省のこころの耳のストレスチェック制度についてというホームページには，ストレスチェックをする上での実施マニュアル，ストレスチェックに対する Q&A，数値基準に基づいた高ストレス者の選定の仕方などストレスチェックに対する様々な項目が記載されている．また，ストレスチェックシートは様々な種類分けがされており職業性ストレス簡易調査票，イラストやふりがなが書いてあるストレスチェック質問票，知的障害等のある労働者のストレスチェック制度実施に関する運用マニュアル，家族視点からストレスを測定する家族による労働者の疲労蓄積度チェックリスト，うつ・不安障害を確認するためのチェックリスト，運輸事業従事者のためのメンタルヘルスこころの健康自己チェックなど様々な人に対してのストレスチェックシートが存在する．

先行研究では新職業性ストレス簡易調査票の職場の資源 (強み) チェックリストを作成した研究がある [19]．この研究のストレスチェックシートは，課題レベル (普段の業務や作業

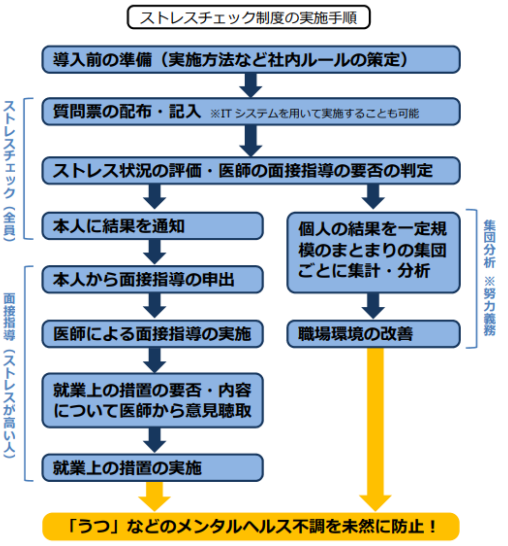


図 3.3: ストレスチェックの流れ

図 3.4: 職業性ストレスチェックシートの一部

に関するもの)，部署レベル (チームや部署の人間関係に関するもの)，事業場レベル (組織のあり方に関するもの) の 3 つの水準に分類し，ストレス対策を行っている．さらにストレスチェックを活用し，職場活性化の参加型グループワークの実施マニュアルも作成している．マニュアルの内容は，60 分間の職場活性化グループワーク (参加型討議) を運営するための手順が記載されている．これは，従業員のメンタルヘルスの向上を図ることが目的である．

また，職場だけではなく医療分野などでの患者に対するストレスチェックシートも存在する [20]．これは，自身の 1ヶ月程度の状態を振り返って回答するものとし，1 (最も良好な回答評定カテゴリー) から 5 (最も不良な回答評定カテゴリー) までの 5 件法の回答を選択するものとしている．これを項目ごとに分析して患者のストレス具合を計測する手法を取っている．

本研究のストレスチェックシートは，富山県立大学の教職員に対して 2021 年度に行われた公立学校共済組合-心のセルフチェックシステムの設問を参考にし，内容を少し変更したものを取り扱うことにした．このストレスチェックシートには質問が全部で 57 問存在する．質問の内容として，研究についての質問を 17 問，気分についての質問が 18 問，体調についての質問が 11 問，周りの支援についての質問が 9 問，満足度についての質問が 2 問という質問構成になっている．このストレスチェックシートを用いて，小型ウェアラブル装置でストレス測定をした期間の自分自身を振り返ってもらい回答してもらうこととする．

§ 3.3　マンマシンシステムにおけるストレスコーピング

マンマシンシステムとは，機械システムとそれを操作する人間とが有機的につなげられた 1 つのシステムのことを指す．マンマシン系は，人間，機械本体，機械と人間の接点であるマンマシンインタフェースから構成される．マンマシンシステムの例としては，航空

機・大規模プラント，自動車・工作機械など日常生活において私たちと関わりの深いものがほとんどである．このような人間の操作により制御されるシステムは，いかに効率的かつ安全なマンマシン系を設計するかが重要となる．図 3.5 には，マンマシンシステムを設計するに当たっての一般的な流れを示す．

マンマシン系の設計で考慮すべき点として，人間と機械システムとの適切な役割分担の設定，人間の指，手，足などに対する適切な操作入力の割り当て，マンマシンインターフェース部機能の適切な設計，すなわち視覚・聴覚・力覚の表示方式の設計，スイッチ・ハンドルといった操作不要の設計，心理的不安感を抱かせない情報フィードバック方式の設計，人間を取り巻く作業環境の把握など様々な点があげられる．

マンマシンシステムにおいて「どのようにすれば，信頼性に低い要素を用いて信頼性の高いシステムを構成することができるか」ということを原点にして，マンマシンシステムの信頼性について言及している論文がある [21]．1960 年代半ば頃から 70 年代前半の考察の中心のシステムとしては，システム構成要素の故障がいくつか重なると，システム故障が発生するという単調性を基盤に置いたコヒレント・システムであった．

しかし，70 年代後半には必ずしも単調性が満たされないシステムを開発する必要性が指摘されるようになった．このようなシステムのことを非コヒレント・システムといい，センサ情報に基づきコンピュータが対象を制御するようなシステムを指す．システム設計において「独自性」や「新規性」を取り入れる時には，システムの信頼性・安全性を考える必要がある．

マンマシンシステムの具体的な例として，原子力プラント，化学プラント，航空機，生産システムなどがあげられる．その中の航空機について最近の航空機におけるマンマシンシステムについて言及している論文が存在する [22]．これまで，民間輸送機技術革新と共に安全性の向上，高速化，大型化，効率化が常に図られてきた．このようなことを考えることと並行に安全性と信頼性も同様に向上させていかなければいけず，航空機の精度が向上するに連れてシステムの多重化や乗員への情報量の増加等を図る必要があった．

航空機がアナログ機器であった時代では，操縦室のシステムの構成が複雑になってしまいスイッチ，表示灯，計器類の数だけでなく，注意警報の種類と数も多くなり，それに伴い操作手順も複雑かつ煩雑になってしまうことが多々あった．

しかし，航空機にディジタル機器が導入されるようになり，これによって離陸から着陸までの飛行の自動化，システム操作とモニタの自動化，および計器の統合化などが可能となった．これによって，スイッチ，表示灯，計器類等の数は減少し，操縦室の計器パネルは高度に洗練された配置，構成，機能となった．また，乗員に対する情報量も減少したため航空機の事故をさらに減少させることができた．図 3.6 には，航空技術の進歩によって航空事故発生率が減少していることを示す．このようなディジタル技術の革新は，飛行機のシステムの自動化だけでなく人間工学面での問題も解決することができる．

また，ストレス計測についてのマンマシンシステムの研究で一過的な作業負荷によって生じる機械操作時のストレスをリアルタイムで評価するシステムを開発した研究がある [23]．マンマシンシステムの設計において，性能や安全性だけでなく，人と機械の親和性を考慮した扱いやすさ，疲れにくさを検討することが不可欠である．

機械操作時の精神的な作業負荷 (メンタルワークロード) は，作業者の主観的評価や行動計測などで評価されている．しかし，作業中に疲労の状態を正確に主観的評価するのは困

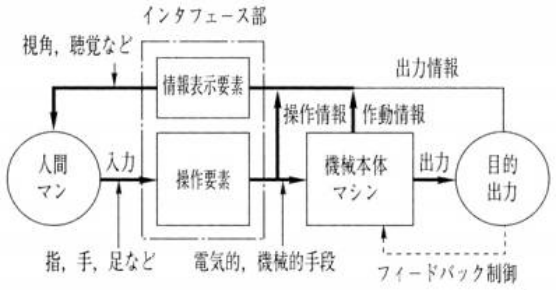


図 3.5: マンマシンシステムの流れ [22]

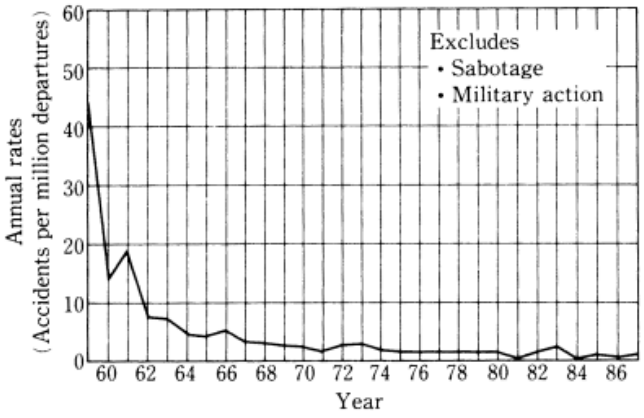


図 3.6: 機体全事故率の推移 (全世界民間ジェット機) [22]

難であり，行動計測においては操作ミスと作業負荷の対応が明確な場合のみしか利用することができない．逆に生理計測では，作業効率と無関係に作業時の負担を連続的，定量的に監視できることより実作業への適用も盛んとなってきている．

生理計測を行うための提案手法として，ディスポーサブル電極を用いて呼吸性洞性不整脈 (Respiratory Sinus Arrhythmia: RSA) を計測し，数十秒分のデータから心拍変動の呼吸周波数成分のパワーとして RSA 振幅を算出し，その RSA 振幅の平均値の変化 (トレンド) から，副交感神経活動の時変特性をリアルタイムにオンラインで評価する．また，オンラインで RSA 振幅のトレンドを推定する手法としてはカルマンフィルタを使用した．実験結果として，提案手法を用いることで副交感神経の状態変化を遅延なく推定できることを示した．

NTT ドコモは慶應義塾大学医学部精神・神経科学教室，東京大学人工物工学研究センターと共同で，スマートフォンを用いることにより自身のストレス状態を推定できるような技術を開発した [24]．この技術を汎用化することを目指すに当たって，アプリケーション (応用ソフト) の早期実用化を目指す．

慶應義塾大学ではストレス状態における行動特性の知見を持っており，また東京大学では行動識別に関するセンサデータ処理の知見を有する．これらの技術とドコモの人工知能 (AI) 技術などを組み合わせることにより開発した．

まず，ストレス状態の客観的な計測方法の 1 つで本研究でも用いているような心拍間隔の揺らぎを用いて解析する．ストレスを定量的に計測することで，利用者のストレス状態を数値化してストレス状態を示す．そしてスマートフォンから得られるセンサデータや位置情報の各種データから移動パターンや電話回数など他者との交流するようなストレス時に現れる行動を約 130 種類の特徴として数値化する．

このストレス値と行動特徴の数値の関係性を AI で学習し，ストレス推定モデルを構築する．これによって，スマートフォンから取得した行動特徴の数値をこのモデルに照らし合わせることによって自身のストレス状態を推定できるといったシステムとなっている．3.7 では，スマートフォンのセンサログデータからの行動特徴の生成，ストレス時における特

図 3.7: ストレス推定モデルの構築

徴的な行動からストレス推定，行動特徴とストレス推定の度合いを表した図を示す.

第4章

提案手法

§ 4.1　生体環境センサの小型化

本研究で使用するウェアラブル装置は，本論文の 2．1 節で示した通り Raspberry Pi Zero WH と Arduino nano を連結した装置を用いることにし，また Arduino nano に 6 つのセンサを配線してそれぞれのセンサ値を取得する．そして，Raspberry Pi Zero WH の方にはマイク入力での行動識別のために Respeaker 2-Mics Pi HAT，カメラでの静止画像取得用に Raspberry Pi Zero 用のカメラを用いた．
この本研究で使用しているウェアラブル装置は，先行研究にて Raspberry Pi 3 (Model B) と Arduino Uno，多種多様なセンサ，USB マイク，Raspberry Pi 用カメラを使用してものを小型化したものとなる．本研究での小型化ウェアラブル装置と先行研究で使用しているウェアラブル装置の大きさはそれぞれ下の図となる．

ウェアラブル装置は，まず Raspberry Pi と Arduino をそれぞれ Raspberry Pi Zero WH と Aruino nano へと変更した．それぞれのマイコンボードは小さくなり，Raspberry Pi Zero WH は USB ポートが少なくなってしまう部分は多少不便ではあるが，無線機能や GPIO の pin 数などにほとんど違いがないためほぼ Raspberry Pi 3 (Model B) と同様の機能を持ち合わせている．また Arduino nano の方も，動作電圧，容量，デジタル入出力などのほとんどのものに大きな違いがないため Arduino Uno と同様に使用することができる．

センサの部分については，先行研究と比較すると GPS センサ，人感センサを取り除いた．まず GPS センサについては，先行研究にて決定木分析より重要なセンサを調べる際に室内，室外のどちらにも重要なセンサとして認識されることがなかったことと GPS センサの用途はストレス測定をした際の場所の記録のために使用されており，これはマイクでの場所と状態を音声入力する部分と少し被ってしまうので今回は取り除くことにした．また人感センサについても GPS センサと同様にマイク入力の行動識別で人と接しているかどうかを判断することができることから人感センサも取り除くこととした．

そして，音声入力用マイクとしては Respeaker 2-Mics Pi HAT を使用した．先行研究では，Raspberry Pi 3 (Model B) だったため USB ポートが 4 つあり USB マイクを使用することができたが，Raspberry Pi Zero WH では micro USB が 1 つしか付いておらずこの部分を Arduino nano との接続に使用するため今回は USB マイクを使うことができなかったため Respeaker 2-Mics Pi HAT を採用した．しかし，Respeaker 2-Mics Pi HAT を使用しても音声識別をすることはでき，さらに Raspberry Pi Zero WH の上にはめることで使用できるので幅を取ることなくウェアラブル装置を作成することができた．

小型化をした最大の理由として，先行研究のウェアラブル装置でストレス測定を行う際

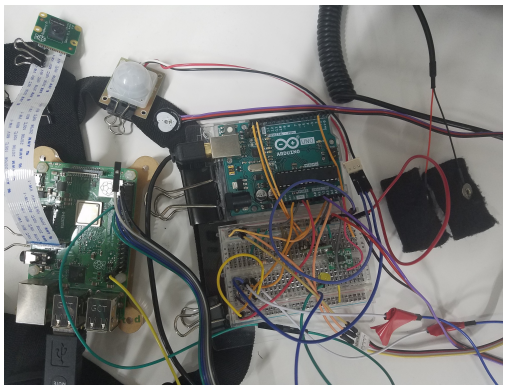


図 4.1: 先行研究のウェアラブル装置

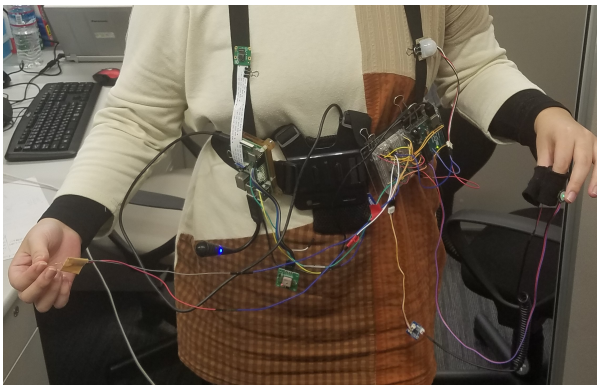


図 4.2: 本研究の小型ウェアラブル装置

にウェアラブル装置を付けていることがそのままストレス値に反映してしまうということがあったためである．これでは，正しいストレス値を算出することができなくなってしまうため，ウェアラブル装置を付けたらストレスがかかってしまうという部分を本研究では最小限に減らした．

また，センサを取り付ける際にはブレッドボードではなくユニバーサル基板にそれぞれのセンサの脚などをはんだ付けしたものを作成した．ブレッドボードだと導線やセンサなどが室外でウォーキングなど動いている時のストレス測定をする場合に外れてしまう可能性があり，センサが外れてしまうことによりそのセンサの値を測定することができなくなりプログラム上でエラーが出てしまい，大変不便だからである．そこでセンサなどが外れてしまうことがなくなるようにユニバーサル基板を用いてはんだ付けを行った．

またユニバーサル基板にはんだ付けをした Arduino nano と様々なセンサと Raspberry Pi Zero WH をそのまま装着するとなると装着することが難しかったので Prusa i3 MK3S+ (図 4.3 参照) という 3D プリンタを使用して独自で Raspberry Pi Zero WH 用のケースとユニバーサル基板にはんだ付けした Arduino nano と様々なセンサが収納できるようなケースをそれぞれ作成した．

先行研究では，コーピング指示を見る手法として最も付け心地が良いという観点から MOVERIO シリーズのスマートグラスを使用していた．スマートグラスだと常に自身の視界の先にコーピング指示が出力されているためコーピング指示を即座に確認できるという利点はある．

しかしその逆で，スマートグラスは AR であり仮想現実にコーピング指示の出力結果を表示するのでこの手法だとコーピング指示が見えにくかったりと正しいストレス緩和が行えない可能性が生じてくる．また，コーピング指示にだけ注意していると室外などでストレス計測を行った場合，人とぶつかってしまうことや身の回りの安全確認が遅れてしまうなど様々な危険な要因が生じてくる．よって室外などでストレス計測を行う時には安全に行う必要があると考えた．

そこで本研究では，先行研究のスマートグラスはコーピング指示の出力結果を常に確認することができるという利点を取り入れることができかつ，身の回りの安全を確認できるイヤホンからのコーピング指示というものを考えた．イヤホンからの音声出力コーピング指示を取り入れることにより，室内でのパソコン作業などにおいてパソコン画面から目を

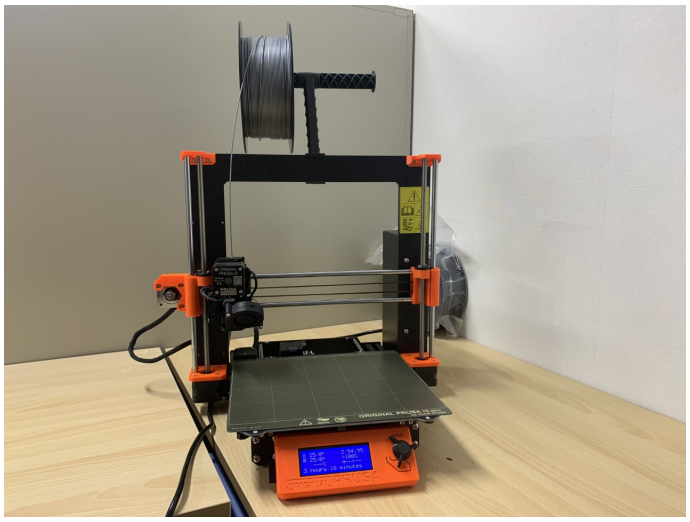


図 4.3: 3D プリンタ

離してしまいコーピング指示に気付かなかったとしても，音声でコーピング指示が聞けるようになるためコーピング指示を見逃してしまうことがなくなる．

また，スマートグラスを取り外したことによって室外でのストレス測定での危険性も無くすことができるため室外での様々な状況下でのストレス測定を行うことが可能となった．

そして，心拍時系列データの波形を時間的にも捉えることを可能にするため本研究では 2.3 節の通り PSD の算出の方法を高速フーリエ変換から連続ウェーブレット変換へと変更した．高速フーリエ変換では周波数成分だけを局在化するが，連続ウェーブレット変換では時間と周波数の両方の成分を局在化することが可能である．ウェーブレット変換はフーリエ変換とは異なり，解析をするためのウィンドウ幅をその点その点においての波形に変化させることができる．よって，非定常な波形において細かい周波数抽出が可能となる．また，マザーウェーブレットには心拍時系列のような信号に向いているガボールウェーブレットをマザーウェーブレットとした．そしてウェーブレット変換をするために様々な値を決定しなければならないが，本研究では Arduino nano から Raspberry Pi Zero WH に心拍センサのデータが 1 秒間に約 3.3 個送信されるのでサンプル周波数を 3.3Hz，これによって時間間隔を 0.3030，ナイキスト周波数を 1.65Hz，スケールを 0.0001 からナイキスト周波数までにしこの範囲を 1024 に分割することで細かな PSD を算出できるようにした．ウェーブレット変換では，パワースペクトルの推定において時間領域を考慮するためパワースペクトルの波形がスケールを 1024 に分割したため 1024 の波形が出現する．しかし，1024 の波形をすべて積分してストレス値を算出しようとするとフーリエ変換では約 30 秒ほどでストレス値を算出していたのが約 5 分ほどになってしまい，短期ストレス計測をすることは難しくなってしまう．今回は，大体同じ時間ぐらいで算出できる 5 つのパワースペクトル配列を抜き出すことにし，200 番目，400 番目，600 番目，800 番目，1000 番目の配列を抜き出すことにした．下記にストレス値を算出する LF/HF の具体的な式を記述する．

$$\frac{LF}{HF} = \frac{LF_1 + LF_2 + LF_3 + LF_4 + LF_5}{HF_1 + HF_2 + HF_3 + HF_4 + HF_5} \tag{4.1}$$

LF_1 と HF_1 はそれぞれ 200 番目のパワースペクトルの LF 成分と HF 成分，LF_2 と HF_2 はそれぞれ 400 番目のパワースペクトルの LF 成分と HF 成分，LF_3 と HF_3 はそれぞれ 600 番目のパワースペクトルの LF 成分と HF 成分，LF_4 と HF_4 はそれぞれ 800 番目のパワースペクトルの LF 成分と HF 成分，LF_5 と HF_5 はそれぞれ 1000 番目のパワースペクトルの LF 成分と HF 成分を表している．

§ 4.2　コーピングの内容決定と音声・画像の出力

本研究での html 表示とイヤホンからの音声出力によるコーピング指示は，Arduno nano でそれぞれのセンサ値を取得した後 Raspberry Pi Zero WH にセンサ値を送信する．Raspberry Pi Zero WH 上では，センサ値に加えてマイクの音声入力したものとカメラで撮影した静止画像を研究室のクラウドサーバに送信するものとする．最後にこれらのデータがサーバ上に送られてきたら，サーバがデータを受信した時間を何月何日何時何分何秒で記録できるようにしサーバ上のセンサデータを記録しておく csv ファイルに時間とセンサデータを格納しておく．
Raspberry Pi Zero WH からクラウドサーバにセンサデータが送られてくる間隔は 1 分毎に設定してあるのでコーピング指示もそれに伴って Raspberry Pi Zero WH からデータが送られてきた時間，ストレス値，コーピングが必要かどうかストレス状態を表す図とコーピング指示の内容も同様に 1 分毎に更新されるようにした．図 4.4 は，自身のストレス状況を簡単に表した図であり色分けをすることによってストレス状況を捉えやすくした．

まず，コーピングの HTML 表示の内容としては，現在時刻，現在地，現在の行動，現在の行動経過時間，ストレス値，ストレス状況，コーピング指示，ストレス状況を示した図，LF/HF のグラフの 9 つの項目である．また，このコーピング HTML 表示はパソコンなどで作業をしている最中にパソコン画面右上などに作業の邪魔にならないような大きさのものを考えている．コーピング指示を画面で見る際に作業中に人間自身が 1 分毎にページの再読み込みをしていると作業の邪魔になってしまい，手間がかかってしまう．このような部分を解決するために Google Ghrome の拡張機能の 1 つである Easy Auto Refresh を用いることとし，コーピングの HTML は 1 分毎に更新するようにしたので 30 秒で繰り返しページを再度読み込めるように設定した．表 4.1 は，コーピングの HTML 出力とイヤホンからの音声出力として出てくる文章のパターンを表したものである．イヤホンから出てくる音声はコーピング HTML で作成されたコーピング指示を簡易化したものになっており，声によってさらに安心させることや画面のみのコーピング出力だと見逃してしまう可能性もあるためこれを無くすといった効果がある．

そして，本研究ではコーピングの HTML 表示の見逃しを防ぐためにウェアラブル装置に付属しているイヤホンからもコーピング指示を出せるようにした．この仕組みは，サーバに送られたセンサデータを基にしてサーバ上で決定木分析を行う．この決定木分析の役割は，たくさんのセンサデータのサンプルを参考にしながらそれぞれのセンサ値が大体どのような値の時にストレス値 5.0 を超えているのかをプログラムで機械学習させ，ストレスを感じている時のそれぞれのセンサのおおよその値を決定する役割である．
そして，決定木分析した結果も可視化できるようにしどのセンサが行動識別を取り入れたストレス計測において関係性が高いのかを示せるようにする．ここでのストレス測定との

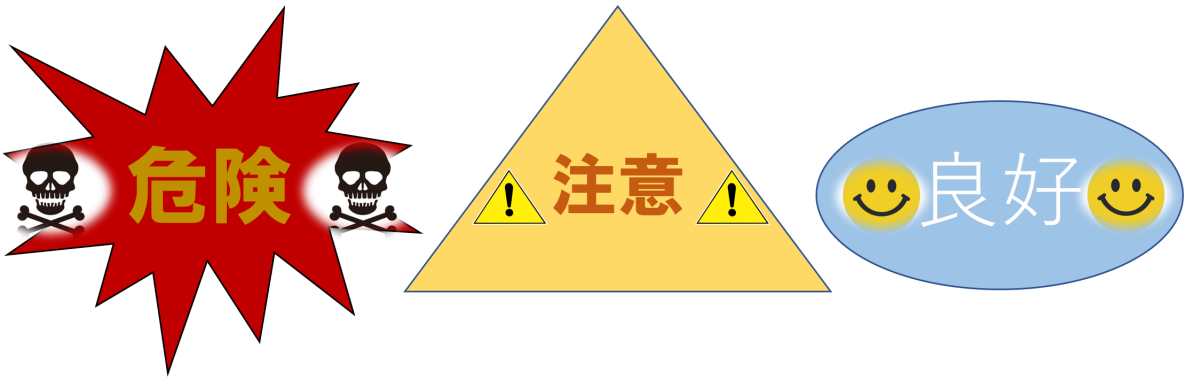


図 4.4: コーピング HTML のイラスト

表 4.1: 行動識別によるコーピング決定表

行動識別	Cope2	Cope3
パソコン	‘コンピュータ操作により’	‘目を休めてください’
会議中	‘会議中なので’	‘深呼吸してください’
運動	‘疲れているので’	‘少し休憩しましょう’
休憩	‘休憩だけど’	‘負荷がかかってます’
…	…	…
不明，他	‘その他の行動により’	‘行動変化をしましょう’

関係性の高いセンサの判別方法は，決定木分析の図においてより上の方のノードに出てきたセンサほどストレス測定との関係性が強いとしている．以下にコーピングする際に必要とする 9 つの項目の詳細を記述する．

現在時刻，現在の行動経過時間

現在時刻は，python のライブラリである datatime モジュールを使用することで現在の時間を取得することとする．また現在の行動経過時間については，クラスター配列を参考にする．これを参考にして，クラスター番号が変わった時刻，マイクで別の行動識別を入力しサーバ上でもその行動識別に変わった時刻を行動開始時刻として，現在時刻から引いた時刻を行動経過時間とする．長時間同じ行動をし続けることは自身に負荷を与えてしまうので，行動経過時間を 45 分を限度としそれ以上は行動を変化させるようなコーピング指示や警告文を表示するものとする．

現在地，現在の行動

現在地と現在の行動については，クラスター配列とマイクのラベルによって識別する．マイクでの行動入力は，行動が変わった際に毎回入力する．入力の仕方は，”研究室 パソコン作業” といったように現在の場所と現在の作業内容を 1 秒ほど間隔を空

けてマイクに入力するようにする．また，マイクラベルを取得する際には周囲の雑音が間違って入力されないようにするため，ホットワードが認識された時のみにマイクラベル用の xlsx ファイルに記録されるようにする．各クラスターには番号が振られているが，マイクラベルにもクラスターごとに番号を振り分ける．これをすることにより，マイクラベルの 1 単語目を現在地，2 単語目を現在の行動として判別することが可能となる．マイク入力をしなかった場合やクラスター分析によってマイクラベルがなかった場合には，現在地，現在の行動の 2 つとも不明となるようにする．

ストレス値，LF/HF のグラフ

ストレス値は，2.3 節の通り PSD から HF 成分と LF 成分の大きさを求めて LF/HF 値を算出する．LF/HF 値は，Raspberry Pi Zero WH 上で Arduino nano から受け取った心拍センサのデータを基にして示す．また，LF/HF のグラフはライフログのデータから最新の LF/HF 値を 20 個 (約 10 分間分) を取得できるようにし，python の matplotlib モジュールを用いてグラフを作成する．作成したら，画像ファイルに変換して HTML 上に表示できるようにする．

ストレス状況，ストレス状況を示した図

ストレス状況として，本研究の判定は 3 種類あり “良好”，“注意”，“要注意” となっている．この判定方法は，それぞれ LF/HF 値が 2.0 未満，2.0 以上 5.0 未満，5.0 以上の 3 つの場合分けによって行っている．今回は，それぞれのストレス状況に応じてコーピング指示のみだけでなくそれぞれのストレス状況を図でも表した．理由としては，コーピング指示のみだと即座に自分自身のストレス状況の把握が難しいため，これを図で表すことによりストレスの把握を簡易的にするためである．これらの画像も LF/HF 値によってそれぞれ相応しい画像に自動的に切り替わるようにする．

コーピング指示

コーピング指示は，現在の行動経過時間，ストレス状況，現在の行動の 3 つから決定する．今回コーピング指示は，HTML 表示とウェアラブル装置のイヤホンからの音声出力の 2 つのパターンがあるので，どちらかを確認してユーザ自身にコーピング指示の内容を行ってもらう．

§ 4.3　提案手法のアルゴリズム

最後に，本研究でのウェアラブル装置による短期ストレス測定とストレスチェックシートによる中長期のストレス測定のアルゴリズムについて述べる．図 4.5 に短期ストレス測定と中長期ストレス測定の提案手法の全体の流れを示す．図 4.5 には含まれていないが，中長期の際に使用するストレスチェックシートの流れは被験者に回答してもらう，全ての合計点と項目ごとの合計点を出す，全ての被験者でのストレス負荷ランキングを出すといった流れになっている．図 4.6 ではウェアラブル装置でのセンサデータの送受信の一連の流れについ

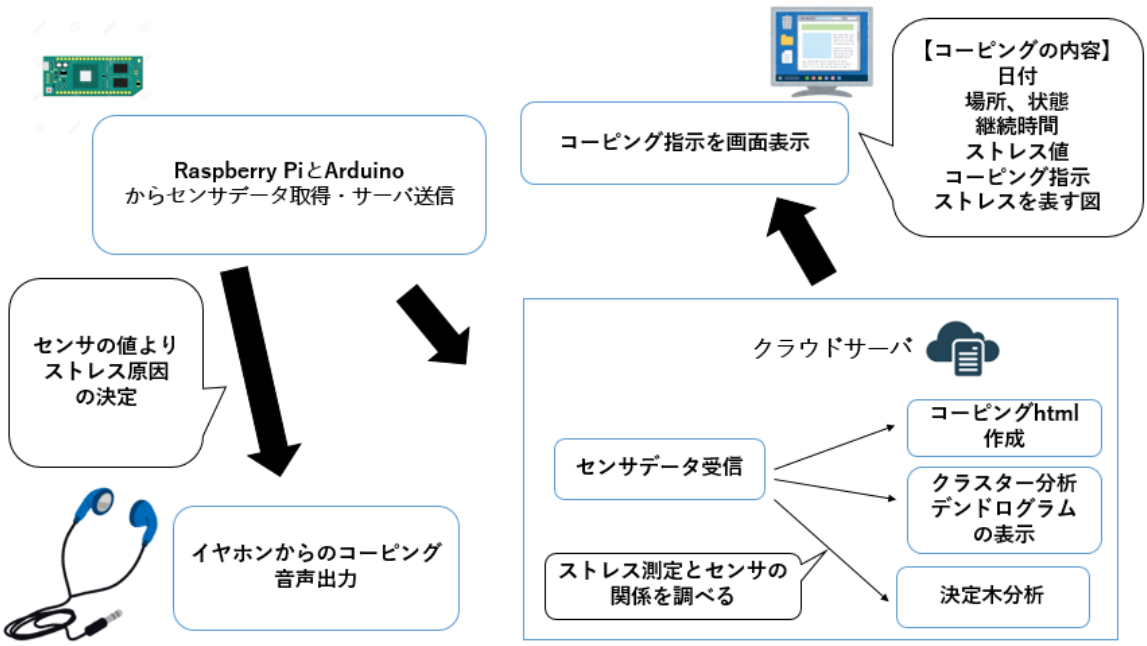


図 4.5: 提案手法の概要

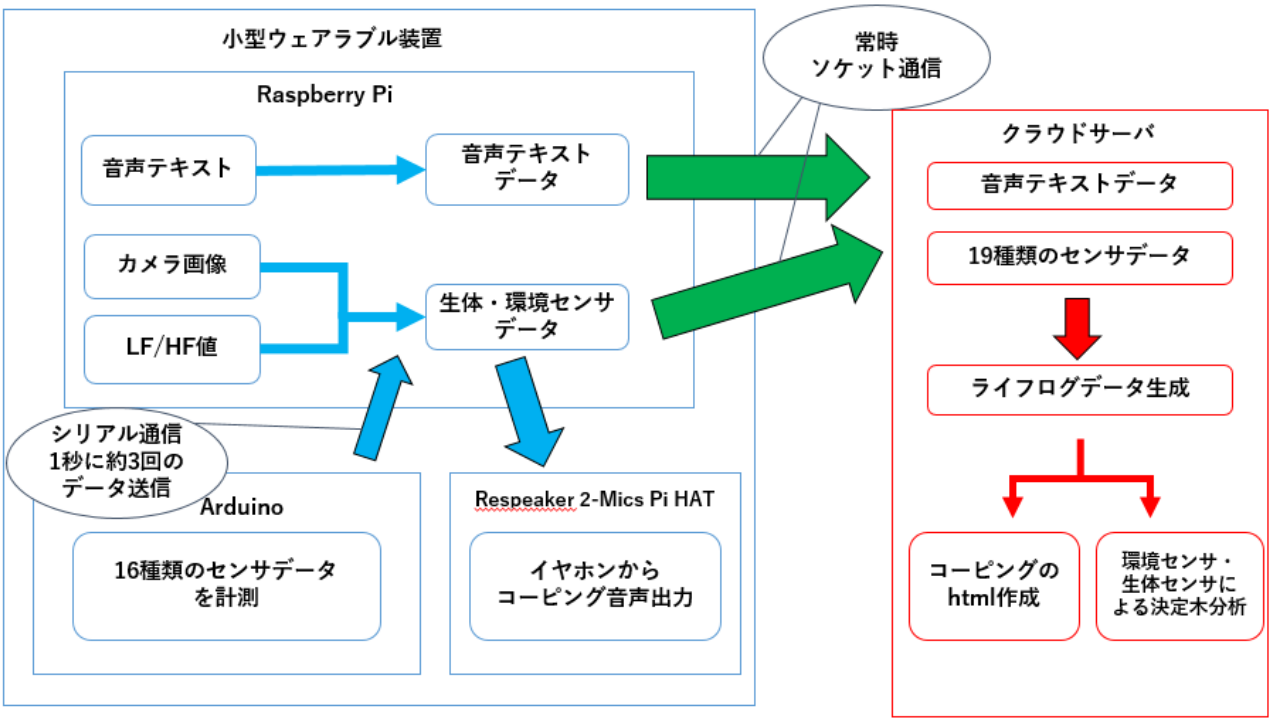


図 4.6: 生体・環境センサデータの流れ

て詳細に表したものを示す.

Step 1 小型ウェアラブルセンサを用いたストレス値の取得

まず，Arduino nanoで温度，湿度，気圧，照度，加速度 (3軸)，角速度 (3軸)，地磁気 (3軸)，体温，心拍，ガルバニック皮膚反応の16種類のデータをシリアル通信によってRaspberry Pi Zero WHに送信する．これらのセンサデータは毎秒約3回取得できるようにしている．一方でRaspberry Pi Zero WHでは，Raspberry Pi用カメラを用いた静止画像とRespeaker 2-Mics Pi HATを用いたマイク音声入力データを収集する．センサデータをシリアル通信により取得している際，心拍データはプログラム上で配列として入ってきたデータをすべて格納しておく．心拍データを1024ためた後，ウェーブレット変換を用いてパワースペクトル密度を求めてLF/HF値を算出する．このときにその他のセンサデータ，静止画像も蓄積しておく．1回目のLF/HF値算出後，心拍データの配列は一番古い100のデータを削除し，新しく100のデータを収集する．これにより，2回目以降のLF/HF値は，約30秒ごとに算出される． 音声データは音声認識API，静止画像は画像認識APIを用いてテキスト化する．

Step 2 クラウドサーバにデータを蓄積

Raspberry Pi Zero WHでは，生体・環境センサデータを蓄積しておりそのデータはjsonファイルに変換して保存している．行動識別をする際に必要な音声データはユーザの行動が変化する毎にマイク入力を行い，先程の生体・環境センサデータとはまた別のjsonファイルにテキスト変換されたものを保存しておく．2つのjsonファイルの内，両方あるいは片方のjsonファイルが作成された段階でRaspberry Pi Zero WH上のデータ送信プログラムよりクラウドサーバにセンサデータがソケット通信で送信される．ストレス測定を行う際は常にソケット通信を行っているためRaspberry Pi Zero WH上のデータ送信プログラムとクラウドサーバのデータ受信プログラムは動かしたままにし，互いのIPアドレスを用いて接続状態にしておく．生体・環境センサデータと音声データは，ファイルデータの大きさでどちらかを識別し，その後それぞれ別のクラウドサーバのxlsxファイルにデータが蓄積されるようになっている．

Step 3 クラスター分析によるコーピング指示の決定

サーバ上に生体･環境センサデータと音声データが送信されたら，サーバ上でもセンサデータを記録しておくためにcsvファイルを作成しその中に記録する．このcsvファイルを基にデンドログラム，時系列クラスター番号配列，コーピング指示の作成を行う．サーバ上のライフログデータを用いてウォード法ユークリッド距離クラスター分析を行うことで，デンドログラムと時系列クラスター番号配列は作成される．クラスター分析は，ライフログデータに音声データが含まれているものを各クラスターの規準とし，その音声テキストをクラスターラベルとする．そして，クラスター分析結果とライフログデータからコーピングのHTML作成を行う．コーピングのHTMLについては，ユーザが再読み込みをせずに自動更新できるようにする．そして，コーピングのHTMLが更新されるのと同時にコーピング指示を見逃さないためにイヤホンからも音声出力が行えるようにする．

Step 4 中長期ストレスのためのストレスチェックシート

ストレスチェックシートについては，2021 年度富山県立大学の教職員向けに行われた公立学校共済組合-心のセルフチェックシステムというストレスチェックシートを参考にし，文章を研究室向けに修正したものを採用した．設問は全部で 57 問あり，ストレスの原因は様々な可能性が考えられるため研究についてのことや自身の体調についてなど分野分けを行ってストレス原因が発見しやすくなっている．また，ストレスチェックにおいて最後にアンケートの部分があったが，質問が被ってしまうことや点数化するのが難しかったため省くことにした．

Step 5 集団の短期ストレスと中長期ストレスの比較

ストレスの集団分析を行うとして，本研究では被験者を 3 人としウェアラブル装置でのストレス測定とストレスチェックシートによるアンケート回答を行ってもらう．まず，ウェアラブル装置でのストレス測定の評価の方法は測定期間の中でどれ程コーピング指示の回数があったのかでストレスの評価を行う．また，ストレスチェックシートの評価方法はそれぞれの質問には 4 つの選択肢があり，選択肢の中で最もストレスがかかっている方から 4 点，3 点，2 点，1 点とする．この点数を 55 問の質問すべてのものを足して合計点数が高ければ高いほどストレスがよりかかっているという評価方法にする．この 2 つの評価方法を用いて 3 人の被験者の短期ストレスと中長期ストレスをどちらとも評価し，それぞれでストレスが高い順に並べ替え順位が一致しているかどうかを確認する．

第5章

数値実験並びに考察

§ 5.1　数値実験の概要

本研究では，被験者を研究室の 3 人とし，短期ストレスの測定において生体・環境センサを用いたライフログデータを作成し，ストレス値の算出，コーピング指示の HTML 作成とその決まったコーピング指示をウェアラブル装置のイヤホンからも流すこと，集団分析によるストレス評価，生体・環境センサを用いたストレス測定の有意性の検証を行う．また，中長期ストレス計測においては 4 件法のストレスチェックシートに回答をしてもらい，質問の点数をすべて足し合わせることを行った．

まず，ウェアラブル装置での数値実験の方法を記述する．今回は，被験者 3 人にそれぞれのライフログデータの収集時間を朝の 9 時から 12 時，昼の 13 時から 17 時とした．長時間ウェアラブル装置を装着することでストレスがかかってしまうことを防ぐため，続けて装着する時間を最大でも 4 時間までにした．行動については，クラスター分析を用いた行動識別を行うため自身の行動が変わる場合には毎回マイクに場所・状態を入力する．今回の実験では，外と中でストレス測定を行うことを考え，外では階段の上り下りや散歩をするなど，また室内では研究室での研究活動や昼休憩の昼食などでストレス測定を行ってみた．

小型化ウェアラブル装置での短期ストレス測定では，Raspberry Pi Zero WH 上で 3 つ，クラウドサーバ上で 2 つ動かすプログラムで一連の流れが形成されている．Raspberry Pi Zero WH 上のプログラムの内容は，クラウドサーバに音声テキストデータと生体･環境センサデータを送信するプログラム，Arduino nano で測定しているセンサデータを Raspberry Pi Zero WH 上で受信して送られてきたセンサデータ，センサデータから LF/HF 値を計算して出力するプログラム，実行するとまずホットワードを入力するよう促し，その後場所と状態を入力させてこれを記録するプログラムの 3 つが存在する．また，クラウドサーバ上のプログラムは Raspberry Pi Zero WH から送られてきた音声テキストデータ，生体・環境センサデータを受け取り Excel ファイルに記録するプログラム，生体・環境センサデータと音声テキストデータからコーピング HTML を作成，マイクラベルを活用したデンドログラムの作成を行うプログラムの 2 つがある．プログラムを実行する手順としては，まずサーバ上の Raspberry Pi Zero WH からのデータを受信するプログラムから実行する．先に Raspberry Pi Zero WH 上のクラウドサーバにデータを送信するプログラムから実行してしまうと，クラウドサーバと接続ができないというエラーが出てしまうためサーバ上の受信プログラムを最初に実行してその後 Raspberry Pi Zero WH 上のデータ送信プログラムを実行する．次に，LF/HF 値を出力するプログラムと音声入力を可能とする 2 つのプログラムを実行し，最後にサーバ上でコーピング HTML 作成などの分析を行ってくれるプロ

表 5.1: 作成されたライフログ (自分のに変える)

2021/2/12 14:27	27.2	31.22	1022.29	518		4.996501	0.320667	0.166355	0.141821	0.143361	0.177407	0.226298		small
2021/2/12 14:31	27.21	30.03	1022.26	508		4.787189	0.257831	0.177686	0.11982	0.14802	0.1652	0.223097		computer
2021/2/12 14:31	27.37	30.63	1022.31	297		4.728996	0.258605	0.168908	0.121068	0.137755	0.164548	0.197007		computer
2021/2/12 14:32	27.44	30.89	1022.27	361		4.56199	0.251884	0.171999	0.116887	0.123762	0.163004	0.195404		ceiling
2021/2/12 14:33	27.51	30.67	1022.27	349	• • •	4.597402	0.258519	0.151797	0.101327	0.126883	0.146939	0.197249		ceiling
2021/2/12 14:33	27.6	29.46	1022.23	192		4.568608	0.243666	0.15159	0.10082	0.116545	0.139695	0.182567	学生室 パソコ	ceiling
2021/2/12 14:34	27.71	30.1	1022.19	194		4.911464	0.205018	0.148535	0.105498	0.119185	0.123348	0.137776		indoor
2021/2/12 14:34	27.83	30.04	1022.34	190		4.889877	0.269603	0.209074	0.151343	0.143641	0.238845	0.232649		person
2021/2/12 14:35	27.9	30.26	1022.27	193		4.22488	0.21014	0.182194	0.121257	0.140466	0.149191	0.164478		indoor
2021/2/12 14:35	27.98	30.29	1022.23	183		4.441177	0.216848	0.175332	0.119697	0.140983	0.150278	0.153221		indoor

グラムを実行する．最後のプログラムを実行するタイミングは LF/HF 値が出力されてからでないとクラウドサーバ上にセンサデータが送信されないため，心拍データを 1024 溜めた後に実行する．

表 5.1 は，本研究で使用しているセンサデータを 1 つの csv ファイルにまとめて記録している一部分である．左から，時間，温度，湿度，気圧，照度，加速度 x 軸方向，加速度 y 軸方向，加速度 z 軸方向，角速度 x 軸方向，角速度 y 軸方向，角速度 z 軸方向，地磁気 x 軸方向，地磁気 y 軸方向，地磁気 z 軸方向，体温，心拍，ガルバニック皮膚反応，LF/HF 値，カメラの静止画像を数値化した furniture 軸，animal 軸，plant 軸，behave 軸，food 軸，appliance 軸，マイクテキスト，カメラテキストという順に記録したものである．

ライフログデータの作成とそれに伴うコーピングの HTML 表示は，実行するプログラムによって一括で行う．実行プログラムでライフログデータの csv ファイルを作成した後，最新のデータを参考にして分析する時間を要するため約 1 分後にコーピングの HTML 表示に反映されるようになっている．

小型ウェアラブル装置で測定したセンサデータをクラウドサーバに送信するためには，Wi-Fi 環境は必須となるので今回は研究室用の Android スマートフォンかあるいは自身のスマートフォンのテザリング環境を使用した．室内では研究室の Wi-Fi に接続することは可能ではあるが，研究室内から外に出た際に一度 Wi-Fi の接続が切れてしまうことでサーバとの接続が切れてしまい，再度プログラムを実行するといった手間がかかってしまうので室内，室外のどちらともスマートフォンによるテザリング機能を使用することにより Wi-Fi 接続が途切れることなく使用できるようにした．

実験の際には，小型ウェアラブル装置を左腕か右腕に身につける．最初プログラムを実行し始めた時は心拍データを 1024 個分ためる必要があるのでストレス値などはすぐには表示されない．約 5 分ほどで心拍データ 1024 個はたまるのでその後は心拍時系列データを基に最初のコーピング HTML 表示が作成され，その後毎回 1 分ほどでコーピング指示が更新される仕組みとなっている．コーピング HTML とイヤホンからのコーピング音声出力によりコーピング指示が発動されたら，速やかに今行っている作業を中断しコーピング指示に従いストレスを緩和させる．図 5.1 には，実験中に表示される HTML の変化を示す．

一方で中長期ストレスのストレス測定は，Google form を用いたストレスチェックシートで行う．ストレスチェックシートの評価方法や問題の作成方法については，本論文の 4.3 節に記載されている内容で行う．また，ストレスチェックシートの問題作成をする際に選択肢の片方だけにストレスがかかっている場合に選択するものを寄せてしまうと適当に選択を

図 5.1: コーピング HTML の変化

表 5.2: 作成されたセンサログ

日	時刻	LF/HF値	コーピング発動	行動	コーピング実行
2022.1.22	18:03:07	6.99	1	パソコン作業	1
2022.1.22	18:04:36	5.9	1	パソコン作業	1
2022.1.22	18:06:33	3.35	0	パソコン作業	0
2022.1.22	18:08:16	5.95	1	パソコン作業	1
2022.1.22	18:09:59	3.77	0	パソコン作業	0
2022.1.22	18:11:42	2.83	0	休憩	0
2022.1.22	18:13:25	1.8	0	休憩	0
2022.1.22	18:14:33	4.58	0	休憩	0
2022.1.22	18:16:16	7.47	1	研究	1
2022.1.22	18:17:59	3.92	0	研究	0
2022.1.22	18:19:42	3.22	0	研究	0

してうまくストレス結果が出ない可能性が生じるため，ストレスがかかっている場合に選択する選択肢を均等に左右に設置するようにした．そしてただ合計点を出すだけではどのようなストレスが生じているのかの発見が難しくなってしまうため，全 57 問ある質問を 7 つの項目に分けてストレス原因がどのようなところにあるのかを発見しやすくし，また短期ストレス測定と比較しやすいようにした．ストレスチェックシートを回答する時間は小型ウェアラブル装置でストレス測定を行った最終日に被験者の 3 人に回答してもらうこととする．

§ 5.2　実験結果と考察

まず，ウェーブレット変換とフーリエ変換で約 30 分ほどストレスの測定を行いストレス値の値変動の比較をしたものを図 5.2 に示す．結果的にフーリエ変換よりもウェーブレット変換の方が数値の変動の幅が広いことから，ウェーブレット変換の方が細かなストレス測定ができていると言える．また，フーリエ変換ではストレス値が上昇しているが，ウェーブレット変換では逆にストレス値が上がっているという場面もあるが，これは時間領域を考慮したことによってよりストレス値を詳細に算出することが可能になったことでフーリエ変換では検知できなかった部分が検知できるようになったからだと考える．

また，フーリエ変換ではパワースペクトルを推定する際にプログラム上のパワースペクトルの配列は 1 つのみしか出てこないが，ウェーブレット変換では時間領域を考慮していることより 1024 のパワースペクトルの配列が算出される．今回は 1024 のパワースペクトルの配列から 4.1 節の通りの 5 つの配列を抜き出してこれらの LF 成分と HF 成分をそれぞれ足し合わせて割り算をするという計算方法を行った．図 5.3 には，ウェーブレット変換におけるパワースペクトルの配列 1024 の内，1 つ，5 つ，10 つを取り出した場合のそれぞれのストレス値を比較したものを示す．結果としては，パワースペクトルの配列 1 つ，5 つ，10 つの場合のどれを取ってもストレス値にはさほど影響することはなかった．これより，パワースペクトルの配列を 1024 使う場合でも 1 つ抜き出して使う場合でもストレス値には大きく影響することがないことが言える．今回は，パワースペクトルの配列の数を大きくすればするほど短期ストレス測定をすることが難しくなってしまうことと 1 つだけ抜き出し

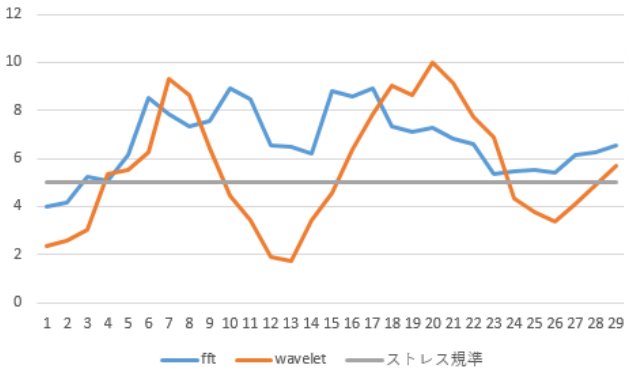


図 5.2: フーリエ変換とウェーブレット変換によるストレス値比較

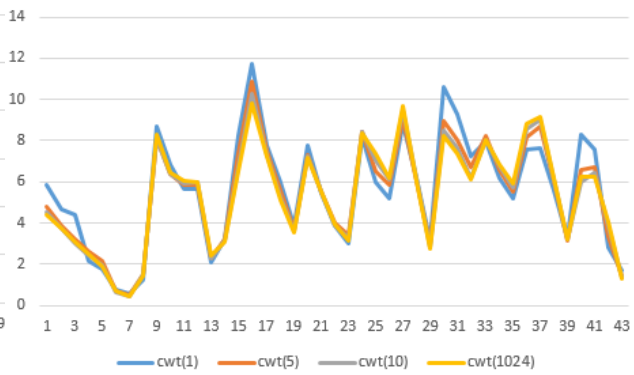


図 5.3: ウェーブレット変換による配列を 4 つの場合で抜き出したストレス値

た場合では時間領域を考慮してるとは言えず，フーリエ変換と同じ計算方法になってしまうので計算のスピードを考慮することから 5 つパワースペクトルの配列を抜き出して計算することにした．

まず，短期ストレス測定においてウェアラブル装置のセンサを用いたストレス測定をしてコーピング指示を発動した際にストレス値が下がっているかを確認する検証を行う．ウェアラブル装置でストレス測定を行った際にコーピングしなかった場合のストレス値の変化を図 5.4 にそれぞれ示す．結果としては，まずコーピング指示に従わなかった場合ではストレス値が上昇した際になかなかストレスが下がらなかったことやストレス値が上昇してしまうことがコーピング指示に従った場合と比較すると多くなるという結果だった．これに対して，コーピング指示に従った場合ではストレス値が上昇した際にコーピング指示に従って少し時間が経過したらストレス値が下がる結果となった．これにより，コーピング指示に従うことでストレスを緩和させることができることが分かる．また，本研究ではコーピングの HTML 作成に加えて小型ウェアラブル装置のイヤホンからもコーピング音声出力を行っているためコーピングを見逃すことをより減らすことができたと思われ，コーピング指示が的確に伝わったこともストレス値を下げることにつながったのではないかと考えている．

短期ストレスの測定と並行に中長期でのストレスチェックシートについても回答を行ってもらい，評価方法も合計点を足す方法で行う．短期ストレスでの小型ウェアラブル装置でストレス測定を行うこととストレスチェックシートによる人間の主観的なストレス測定を行うことによってウェアラブル装置でのストレスコーピングに有効性があるのかを確認することができる．被験者 3 人をそれぞれ A，B，C と設定し小型ウェアラブル装置でストレス測定を場合のストレス値の変化の様子と中長期ストレス測定における項目ごとの点数の割合をレーダーチャートに表したものを図 5.5 から図 5.10 に示す．そして，ストレスチェックシートの分野ごとの合計点数とそれらを全て足し合わせた総計を表にまとめたものを表 5.3 に示す．

まず，図 5.5，図 5.7，図 5.9 の 3 つの図の小型ウェアラブル装置でのストレス測定の結果は，測定期間中のストレス値が 5.0 以上になった際に発動されるコーピング回数でストレス負荷がどの程度かかっているかを決めた結果，A，B，C という順番になった．被験者の A

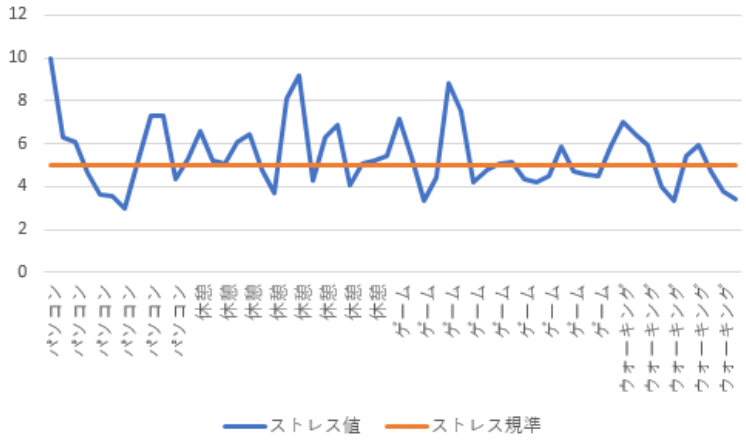


図 5.4: 小型ウェアラブル装置でコーピングしなかった場合

は，学部生の 4 年生であり被験者 B，C はそれぞれ学部生の 3 年生であるため結果は学部の 4 年生である被験者 A が一番ストレスが高く，次に B，C でストレスが高いという結果になった．この結果は，4 年生の人は常に研究活動を続けており，3 年生の人と比較すると土日に学校に来ることがあることや夜遅くまで研究活動を行っていることや先生との打ち合わせにより日々ストレスが溜まってしまう状況が長く続いていることから一番合計点数が高くなり，ストレスを感じているという結果が出たのではないかと考えている．

そして表 5.3 は，中長期のストレス測定におけるストレスチェックシートを回答してもらい，それぞれの項目ごとの点数それらを全て足し合わせた合計点数の結果を表した表である．本研究のストレスチェックシートでは，項目を 7 つに分けている．研究 1 は，現在の研究の量や状況を表しており，研究 2 は研究室の雰囲気や研究室の仲の良さなどを表している．気分 1 では，自身の最近の感情について質問しており，気分 2 では研究に対してどのような感情が現れているかを質問している．体調では，自身の身体で最近悪くなってしまった部分はないかということを質問している．支援では，自身の周りの人は自身に対してどのような接し方をしてくれるかを質問している．最後に満足度では，現在の自身の環境の満足度を質問している．

この結果は A，B，C という順番になり小型ウェアラブル装置でストレス測定を行った際のストレス負荷の順位と一致する結果となった．また，図 5.6，図 5.8，図 5.10 のそれぞれの項目ごとのレーダーチャートはそれぞれの項目での合計点数でレーダーチャートを作成してしまうと項目ごとの質問数が違うため例えその項目で満点であったとしても質問数が少ない項目だと低いように見えてしまう．そこで割合でそれぞれの項目ごとの点数状況をレーダーチャートにすることにより点数状況が把握しやすいように作成した．

次に本研究の小型化ウェアラブル装置でのストレスコーピングに有効性があることを示すために検定を行う．今回検定のために用いるデータはコーピングを行わない場合と行う場合のどちらとも被験者の A とするので対応のあるデータの場合で検定を行っていく．今回の被験者 A のコーピングを行った場合 (図 5.4 参照) ではストレス値が 5.0 を超えたのは 12 回，行わなかった場合 (図 5.5 参照) では 10 回でありこれらのストレス値の変化をまとめ

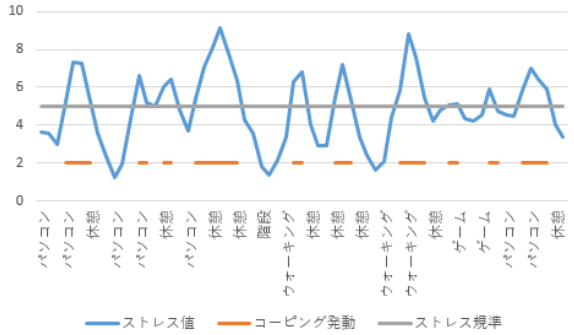


図 5.5: 被験者 A の短期ストレス結果

図 5.6: 被験者 A の中長期ストレス結果

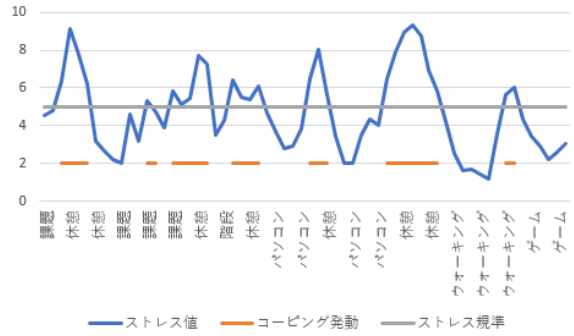


図 5.7: 被験者 B の短期ストレス結果

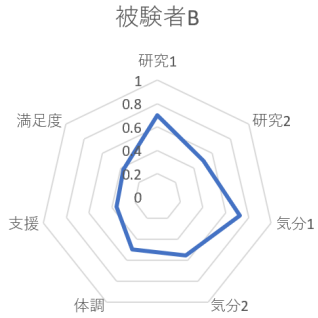


図 5.8: 被験者 B の中長期ストレス結果

たものを表 5.4 に示す．表 5.4 のそれぞれのストレス値の前と後の時間差は約 3 分である．

この表 5.4 を基にして，まず F 検定を行う．F 検定を行うことでコーピングなしとありの場合の分散の検定を行うことができる．変数 1 をコーピングなしの LF/HF 値の差分，変数 2 をコーピングありの場合の LF/HF 値の差分とし，有意水準は $\alpha = 0.05$ に設定した．F 検定の結果は，表 5.5 の通り p 値が 0.192327 となり α を上回る結果となったのでこの 2 つの変数には等分散性がないと言える．

次に等分散性がないことがわかったのでこの場合の T 検定を行う．T 検定を行うことで 2 つの変数の母平均に有意差があるかどうかの検証ができる．帰無仮説 H_0 として “コーピングなしの場合の LF/HF 値の差分とコーピングありの場合の LF/HF 値の差分には差があるとはいえない” とする．実際に T 検定を行い有意水準と比較するための p 値を算出した結果，p=0.002754 となり，α 値よりも p 値の方が下回る結果となったので帰無仮説 H_0 は棄却される．これにより，コーピングなしの場合とありの場合での LF/HF 値の差分には有意な差があることが説明できる．これらの結果はまとめて表 5.5 に示す．

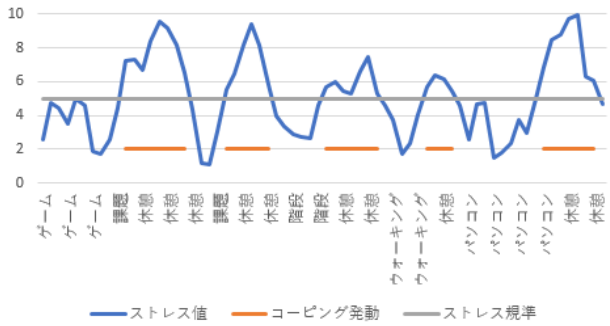


図 5.9: 被験者 C の短期ストレス結果

図 5.10: 被験者 C の中長期ストレス結果

表 5.3: ストレスチェックシートの合計点数

項目	研究 1	研究 2	気分 1	気分 2	体調	支援	満足度	総計
被験者 A	35	15	24	26	19	19	3	141
被験者 B	28	14	26	20	22	13	3	126
被験者 C	27	14	20	19	14	16	4	114

表 5.4: LF/HF 値の差分データ

コーピングしない場合			コーピングする場合		
前	後	差	前	後	差
6.3	6.1	0.2	7.27	5.25	2.02
5.25	7.3	-2.05	6.61	5.19	1.42
5.25	6.61	-1.36	6.05	6.42	-0.37
5.08	6.05	-0.97	9.15	6.35	2.8
8.1	9.15	-1.05	6.83	4.04	2.79
6.3	6.83	-0.53	7.17	5.32	1.85
5.05	5.21	-0.16	7.55	5.47	2.08
8.81	7.55	1.26	5.89	4.71	1.18
5.06	5.12	-0.06	5.85	6.99	-1.14
5.89	4.71	1.18	6.43	5.9	0.53
5.85	6.99	-1.14			
5.39	5.95	-0.56			

表 5.5: LF/HF 値の差分の検定

F 検定		
	コーピングなし	あり
平均	0.436667	-1.316
分散	0.977333	1.695671
観測数	12	10
P 片側	0.192327	

T 検定		
	コーピングなし	あり
観測数	12	10
t	3.498263	
P 片側	0.001377	
t 境界値 片側	1.739607	
P 両側	0.002754	
t 境界値 両側	2.109816	

第6章

おわりに

本研究では，小型ウェアラブル装置でのストレス測定とストレスチェックシートによるアンケート回答をそれぞれ行い，短期ストレスと中長期ストレスの状態は一致するのかを検証した.

短期ストレスのウェアラブル装置でのストレス測定は，ウェーブレット変換を用いて時間軸を考慮したため複数の RRI 波形が作成された上での 1 つずつの RRI 波形の HF 成分，LF 成分のパワースペクトル密度の推定を行い，LF/HF 値を算出することができた．そして，コーピング指示の部分ではデスクワークなどにおいて作業中の画面に邪魔にならないようにコーピング html 表示とパソコン画面を見ていなかった際のコーピング指示見逃しを防ぐためにウェアラブル装置のイヤホンからのコーピング指示音声出力の 2 つのパターンでコーピング指示を行った.

また中長期ストレスでのストレスチェックシートのアンケート回答は，公立学校共済組合-心のセルフチェックシステムのストレスチェックシートを参考にして文章を研究室バージョンに変えたものを参考にした．そして，すべての質問には 4 つの選択肢を用意してストレスがかかっている方から 4 点，3 点，2 点，1 点とし質問すべての合計点を足すことでストレス状況を点数化することができた.

実際に短期と中長期のストレス測定を行い，短期ストレスについてはウェアラブル装置でストレス測定をした際に合計で何回コーピング指示をされたかでストレス状況を判定することにし，また実際にコーピング指示を行った際にストレス値が下がったことを証明するため，コーピングを行わない場合も計測してみた．中長期のストレス測定では，ストレスチェックシートに回答をしてもらった上で選択肢ごとに点数を決めてそれらをすべて足し合わせるということでストレス状況を判定した.

そして，短期と中長期でのストレス状況をそれぞれでランキング付けをして実際にストレス負荷順位を比較して○○が得られた.

謝辞

本研究を遂行するにあたり，多大なご指導と終始懇切丁寧なご鞭撻を賜った富山県立大学工学部電子・情報工学科情報基盤工学講座の奥原浩之教授，António Oliveira Nzinga René 講師，電子・情報工学科 3 年の北田真悟さんに深甚な謝意を表します．最後になりましたが，多大な協力をして頂いた，研究室の同輩諸氏に感謝致します．

2022 年 2 月

瀧田　孔明

参考文献

[1] “アンビエント社会とは”，https://k-tai.watch.impress.co.jp/docs/column/keyword/303757.html, 閲覧日 2021.12.1.

[2] 生活習慣病予防協会,“座ったままの生活の死亡リスク”，http://www.seikatsusyukanbyo.com/calendar/2019/009893.php, 閲覧日 2021.12.2.

[3] 岩倉成志, 西脇正倫, 安藤章, “長距離トリップに伴う運転ストレスの測定-AHS の便益計測を念頭に-”, 土木計画学研究・論文集 Vol.18, No. 3, pp. 439-444 2010.

[4] 佐久間大輝, 神田尚子, 吉見真聡, 吉永努, 入江英嗣, “座位状態での心拍測定を用いたリアルタイムなストレス緩和システム”, マルチメディア, 分散, 協調とモバイル (DICOMO2013) シンポジウム, 2013.

[5] 櫻井 美咲, 矢島邦昭, “生体情報によるストレス計測・分析システムの検討-コンピューターベース学習環境において-”, 情報処理学会東北支部研究報告, Vol. 2016-7-A1-4, 2017.

[6] 新谷隆彦, “ライフログを支える技術”，知能と情報 (日本知能情報ファジィ学会誌), Vol.26, No. 2, pp. 51-56, 2014.

[7] 江崎菜々, “アンビエントコンピューティングによる行動とストレス検知に基づくコーピング支援”, 富山県立大学学位論文 2021.

[8] “コーピングとは?意味や導入法を理解しストレスと上手に向き合おう”, https://survey.lafool.jp/mindfulness/column/0104.html, 閲覧日 2021.12.10.

[9] 高屋正敏, 長谷川泰隆 “ストレスコーピング特性と職業性ストレス”, 産業衛生額雑誌, pp. 807-813, 2016.

[10] Fares, Murhaf; Kutuzov, Andrei; Oepen, Stephan & Velldal, Erik (2017). Word vectors, reuse, and replicability: Towards a community repository of large-text resources, In Jörg Tiedemann (ed.), Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017. Linköping University Electronic Press. ISBN 978-91-7685-601-7.

[11] 伊藤克人, “産業現場でのストレスチェックの実際”, 心身医学, Vol. 56, No.8, pp. 807-813, 2016.

[12] 山口昌樹, “唾液を用いたストレスの計測と回復支援”, 精密工学会誌 Vol. 82, No.8, pp. 731-734, 2016.

[13] 野澤昭雄, 三澤裕樹, 水野 統太, 田中久弥, 井出英人 “顔面熱画像解析による会話形態に関する運転者のメンタルワークロードの評価”, 情報処理学会, Vol.126, No.8, pp.412-418, (2006)

[14] “ストレスと自律神経の科学”,http://hclab.sakura.ne.jp, 閲覧日 2021.12.16.

[15] 横山清子, 森本陽子, 水野康文, 高田和之, “ウェーブレット変換によるパワースペクトル推定法”

[16] “ストレスチェック制度について”, https://kokoro.mhlw.go.jp/etc/kaiseianeihou/, 閲覧日 2021.12.19.

[17] “自分だけではなく身近な人のストレスチェックのポイント”, https://doctorsfile.jp/h/178060/mt/1/, 閲覧日 2022.1.16

[18] 西山 高史, 仲島 了治, 中原 智治, 一見市 伸裕, 椹木 哲夫, “アンビエントテクノロジーの住宅分野への応用の試み”, システム制御情報, Vol. 56, No. 1, pp. 21–26, 2012.

[19] 島津明人, “ストレスチェックの集団分析と職場環境の改善：ストレスチェックの戦略的活用に向けて”

[20] 吉津紀久子, 東井申雄, 白神美知恵, 植園法子, 前川佳敬, 樂木宏実, 永田勝太郎 “患者のストレス早期発見チェックシートの開発”,

[21] 稲垣敏之, “ヒューマンシステム：高信頼性が損なう安全性”

[22] 笹田栄四郎, “最近の航空機におけるマン・マシンシステム”

[23] 松村雄一, 栗田裕, 西小路拓也, “心拍の呼吸性変動を用いた機械操作時の一過性ストレスのオンライン推定法”

[24] “スマホでストレス推定　ドコモ、慶大・東大と開発”, https://www.nikkan.co.jp/articles/view/467293, 閲覧日 2022.1.23

付録

A. 1　生体・環境センサデータを送信するソースコード

生体・環境センサデータを取得し保存するソースコード A.1 を示す.

ソースコード A. 1: sensa-all.py

```
# −*− Coding: utf−8 −*−
import serial
import time
import socket
import requests
import argparse
import json
import cv2

import pandas as pd
import numpy as np
from scipy import signal
from micropyGPS import MicropyGPS
import threading
gps = MicropyGPS(9, 'dd')

import picamera
import picamera.array
with picamera.PiCamera() as camera:
    camera.resolution = (400,300)
    camera.start_preview()
    camera.capture('test.jpg')

import matplotlib
matplotlib.rcParams['backend']='TkAgg'
import matplotlib.pyplot as plt
from PIL import Image
from io import BytesIO
import errno

import collections as cl
import datetime
from timeout_decorator import timeout, TimeoutError

#ikaTEXT
subscription_key = "30fb2bdf2ec54da0a7aa197631fb7848"
assert subscription_key

vision_base_url = "https://japaneast.api.cognitive.microsoft.com/vision/v2.0/"
analyze_url = vision_base_url + "analyze"
image_path = "/home/pi/test.jpg"

parser = argparse.ArgumentParser()
parser.add_argument('−−ip_addr', help="Put the client's ip address here")
args = parser.parse_args()
```

```
46
47
48 connorclo=0
49 def rungps():
50     s = serial.Serial('/dev/serial0', 9600, timeout=10)
51
52     s.readline()
53     t=0
54     while True:
55         sentence = s.readline().decode('utf-8')
56         if sentence[0] != '$':
57             continue
58         for x in sentence:
59             gps.update(x)
60
61
62 def main():
63     RR=[]
64     sin=[]
65     gpio_seri = serial.Serial('/dev/ttyACM0', 115200, timeout=1)
66     dt=1
67     fs = 1.0/dt
68
69     cnt = 0
70
71     print("START")
72     t=1
73     senddic={}
74     print("START")
75     t=1
76     gpsla=gpsla1=0.0
77     gpslo=gpslo1=0.0
78     while True:
79
80
81
82         bio_data = gpio_seri.readline().rstrip()+","+str(gpsla)+","+str(gpsla)
83
84         bd=bio_data.split(",")
85         if len(bd)>17:
86             sin.append(int(bd[15]))
87             if int(bd[15])>0:
88                 r = 60/float(bd[15])
89                 RR.append(r)
90
91
92         cnt +=1
93         if cnt % 10 ==0:
94             print(cnt)
95
96         if(len(RR)>=1024):
97             #thread for GPS
98             gpsthread1 = threading.Thread(target=rungps, args=())
99             gpsthread1.daemon = True
100            gpsthread1.start()
101
102            #mae ha 5
103            dt_now=datetime.datetime.now().strftime('%Y/%m/%d_%H:%M:%S')
104            print(dt_now)
105            print("ondo: "+bd[0]+" sitsudo: "+bd[1]+" taikiatsu: "+bd[2]+"
                   kido: "+bd[3]+" dkenti: "+ bd[4])
```

```
106             print("kasokux: "+bd[5]+" ky: "+bd[6]+" kz: "+bd[7]+" kakusokux:
                    "+bd[8]+" kky: "+bd[9]+" kkz: "+bd[10]+" tizix: "+bd[11]+"
                    tiziy: "+bd[12]+" tizix: "+bd[13])
107             print("taion: "+bd[14]+" sinpaku: "+bd[15]+" GSR: "+bd[16])
108             print(sum(sin)/len(sin),"RRheikin",sum(RR)/len(RR),int(len(RR)))
109             s = pd.Series(RR)
110             freq, pow_den=signal.periodogram(s,fs)
111             vlf = 0.04
112             lf = 0.15
113             hf = 0.4
114             Fs = 250
115             dy=1.0/Fs
116
117             lf_freq_band = (freq >= vlf) & (freq <= lf)
118             hf_freq_band = (freq >= lf) & (freq <= hf)
119             LF = np.trapz(y=abs(pow_den[lf_freq_band]), x=None, dx=dy)
120             HF = np.trapz(y=abs(pow_den[hf_freq_band]), x=None, dx=dy)
121             LF_HF = float(LF) / HF
122             print(LF_HF)
123
124             bio_data=bio_data+","+str(LF_HF)
125
126             with picamera.PiCamera() as camera:
127                 camera.resolution = (400,300)
128                 #camera.start_preview()
129                 camera.capture('test.jpg')
130             image_data = open(image_path, "rb").read()
131             img = cv2.imread(image_path)
132             #cv2.imshow("Flame", img)
133             cv2.waitKey(1000)
134             headers = {'Ocp-Apim-Subscription-Key': subscription_key,
135         'Content-Type': 'application/octet-stream'}
136             params = {'visualFeatures': 'Description'}
137             try:
138                 response = requests.post(
139                     analyze_url, headers=headers, params=params, data=image_data)
140             except requests.exceptions.ConnectionError:
141                 continue
142
143             response.raise_for_status()
144             analysis = response.json()['description']['tags']
145             jsonana=json.dumps(analysis).replace("\"","")
146             print("what?")
147             #send data to MYPC to socket
148             if connorclo==1:
149                 jsonana1=json.dumps(analysis)
150                 gpsthread3 = threading.Thread(target=socket_data, args=(bio_data,
                        jsonana1))
151                 gpsthread3.daemon = True
152                 gpsthread3.start()
153
154             #send data to ss
155             dt_now=datetime.datetime.now().strftime('%Y/%m/%d_%H:%M:%S')
156             senddic["key"+str(t)] = {"nowtime":dt_now}
157             senddic["key"+str(t)].update({"bio_data":bio_data})
158             senddic["key"+str(t)].update({"camera":jsonana})
159
160             t+=1
161             print("roop: "+str(t))
162             print("active: "+str(threading.activeCount()))
163             print(connorclo)
```

```
164
165             if(t==3):
166                 json_data = json.dumps(senddic).replace("\\", "").replace("[","").
                        replace("]","").replace("  ","")
167
168
169                 file_path = "./sendata2.json"
170                 with open(file_path, 'w') as f:
171                     json.dump(json_data, f,ensure_ascii=False)
172
173                 t=1
174                 sendic={}
175                 print("wait")
176
177             if cv2.waitKey(1) & 0xFF == ord('q'):
178                 break
179             cnt=0
180             del RR[:100]
181     cv2.destroyAllWindows()
182
183 if __name__ == '__main__':
184     main()
```

A. 2　マイクのラベルを送信するソースコード

マイクのラベルを取得し保存するソースコード A.2 を示す.

ソースコード A. 2: gasmic.py

```
1  # −*− coding: utf−8 −*−
2  import speech_recognition as sr
3  from datetime import datetime
4
5  import RPi.GPIO as GPIO
6  GPIO.setmode(GPIO.BOARD)
7  GPIO.setup(11, GPIO.OUT)
8
9  import time
10 import json
11 import socket
12 import openpyxl
13 senddic={}
14 import requests
15 import datetime
16
17 r = sr.Recognizer()
18 mic = sr.Microphone()
19
20 while True:
21     print("Start recognized by Ok Google ...")
22     GPIO.output(11, True)
23     with mic as source:
24         r.adjust_for_ambient_noise(source)
25         audio = r.listen(source)
26     print ("Now to recognize it...")
```

```
27      GPIO.output(11, False)
28
29      try:
30          print(r.recognize_google(audio, language='ja-JP'))
31          GPIO.output(11, False)
32          if r.recognize_google(audio, language='ja-JP') == u"ストップ" :
33              GPIO.output(11, False)
34              print("end")
35              break
36          if r.recognize_google(audio, language='ja-JP') == u"Ok Google" :
37              for i in range(5):
38                  GPIO.output(11, True)
39                  time.sleep(0.05)
40                  GPIO.output(11, False)
41                  time.sleep(0.05)
42              print("Say something ...")
43              GPIO.output(11, True)
44              with mic as source:
45                  r.adjust_for_ambient_noise(source)
46                  audio = r.listen(source)
47
48              print ("Now to recognize it...")
49              GPIO.output(11, False)
50
51              print(r.recognize_google(audio, language='ja-JP'))
52              #seikou sitara 2kaitenmetsu
53              for i in range(2):
54                  GPIO.output(11, True)
55                  time.sleep(0.05)
56                  GPIO.output(11, False)
57                  time.sleep(0.05)
58
59
60              dt_now=datetime.datetime.now().strftime('%Y/%m/%d_%H:%M:%S')
61              senddic["mickey"] = {"nowtime":dt_now}
62              senddic["mickey"].update({"mic":r.recognize_google(audio, language='ja-JP
                    ').encode('utf-8')})
63              json_data = json.dumps(senddic)
64              print(json_data)
65              print('send')
66              file_path = "./sendata.json"
67              with open(file_path, 'w') as f:
68                  json.dump(json_data, f,ensure_ascii=False)
69
70              GPIO.output(11, False)
71
72      except sr.UnknownValueError:
73          print("could not understand audio")
74      except sr.RequestError as e:
75          print("Could not request results from Google Speech Recognition
                service; {0}".format(e))
```

A. 3　サーバーにデータを送信するソースコード

サーバーにデータを送信するソースコード A.3 を示す.

ソースコード A. 3: sendata.py

```
1  # −*− coding: utf−8 −*−
2  import speech_recognition as sr
3  from datetime import datetime
4
5  import RPi.GPIO as GPIO
6  GPIO.setmode(GPIO.BOARD)
7  GPIO.setup(11, GPIO.OUT)
8
9  import time
10 import os
11 import json
12 import socket
13 import openpyxl
14 senddic={}
15 import requests
16 import datetime
17
18 client=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
19 client.connect(("133.55.115.240", 80))
20
21
22 while True:
23
24
25     if os.path.isfile("sendata.json"):
26         with open('sendata.json') as f:
27             json_data=json.load(f)
28             client.sendall(json_data)
29             print('send mic')
30
31         os.remove('sendata.json')
32
33     if os.path.isfile("sendata2.json"):
34         with open('sendata2.json') as f:
35             json_data2=json.load(f)
36             client.sendall(json_data2)
37             print('send sensa')
38
39         os.remove('sendata2.json')
40     else:
41         print('wait')
42         time.sleep(10)
```

A. 4 サーバーでセンサデータを取得し，xlsx ファイルに保存するソースコード

サーバーでセンサデータを取得し，xlsx ファイルに保存するソースコード A.4 を示す.

ソースコード A. 4: server.py

```
1  #−*− coding: UTF−8 −*−
2
3  import json
```

```
4  import time
5  from datetime import datetime
6
7  import socket
8  import pandas as pd
9
10 import glob
11 import numpy as np
12 from oauth2client.service_account import ServiceAccountCredentials
13 import nltk
14 from nltk.stem.wordnet import WordNetLemmatizer
15 lemmatizer = WordNetLemmatizer()
16 import folium
17 from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
18
19
20 rcParams['font.family'] = 'sans-serif'
21 rcParams['font.sans-serif'] = ['Hiragino Maru Gothic Pro', 'Yu Gothic', 'Meirio', 'Takao',
       'IPAexGothic', 'IPAPGothic', 'VL PGothic', 'Noto Sans CJK JP']
22
23
24
25 try:
26     with socket.socket(socket.AF_INET,socket.SOCK_STREAM) as s:
27         s.bind(('133.55.115.240',80))
28         s.listen(1)
29
30         while True:
31             conn, addr=s.accept()
32             with conn:
33                 while True:
34                     print("stay")
35                     #サーバーにきたデータをまずxlsxファイルにかく(スプレッドシート
                           みたいなの作成)
36                     data=conn.recv(1024)
37                     if not data:
38                         time.sleep(5)
39                         break
40
41                     print("get data")
42
43                     data=data.decode()
44                     Sheet1=pd.read_json(data)
45
46                     print(len(data))
47
48
49                     #入ってきたのがセンサだった場合
50                     if len(data)>200:
51
52                         ile= pd.ExcelFile('fordend.xlsx')
53
54                         dr1 = ile.parse('Sheet1', index=False,header=None)
55
56                         df = pd.DataFrame([[str(Sheet1.key1.nowtime),str(Sheet1.key1.
                               bio_data),str(Sheet1.key1.camera)],[str(Sheet1.key2.nowtime)
                               ,str(Sheet1.key2.bio_data),str(Sheet1.key2.camera)]])
57                         df_concat = pd.concat([dr1, df],axis=0)
58                         print(df_concat)
59
60                         df_concat.to_excel(ile,index=False, header=False)
```

```
61
62                          #入ってきたのがセンサだった場合
63                          else:
64                              ile2=pd.ExcelFile('fordend2.xlsx')
65
66                              dr2 = ile2.parse('Sheet1', index=False,header=None)
67
68                              df = pd.DataFrame([[str(Sheet1.mickey.nowtime),str(Sheet1.
                                    mickey.mic)]])
69                              df_concat = pd.concat([dr2,df],axis=0)
70                              print(df_concat)
71
72                              df_concat.to_excel(ile2,index=False, header=False)
73
74
75
76
77          #os.remove('fordend.csv')
78
79  except KeyboardInterrupt:
80      print('!!FINISH!!')
```

A. 5　クラスター分析とコーピング処理を行うソースコード

クラスター分析とコーピング処理を行うソースコード A.5 を示す.

ソースコード A. 5: processing-data.py

```
1  #−*− coding: UTF−8 −*−
2
3  import json
4  import time
5  from datetime import datetime
6  import webbrowser
7  import random
8
9  import stress_decision_tree
10
11 import openpyxl
12 import os
13 import socket
14 import pandas as pd
15 import codecs
16 import math
17 import csv
18 import glob
19 import numpy as np
20 from oauth2client.service_account import ServiceAccountCredentials
21 import gspread
22 import nltk
23 from nltk.stem.wordnet import WordNetLemmatizer
24 lemmatizer = WordNetLemmatizer()
25
26 import gensim.models.keyedvectors as word2vec
27 embed_map = word2vec.KeyedVectors.load_word2vec_format("model.bin", binary=True)
```

```
28
29 import folium
30 import webbrowser
31
32 from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
33 from matplotlib import pyplot as plt
34 from matplotlib import rcParams
35
36
37 rcParams['font.family'] = 'sans-serif'
38 rcParams['font.sans-serif'] = ['Hiragino Maru Gothic Pro', 'Yu Gothic', 'Meirio', 'Takao',
       'IPAexGothic', 'IPAPGothic', 'VL PGothic', 'Noto Sans CJK JP']
39
40
41
42
43 names=['時間','温度','湿度','大気圧','輝度','動体検知','加速度x','加速度y','加速度z','角速度
       x','角速度y','角速度z','地磁気x','地磁気y','地磁気z','体温','心拍数','GSR','緯度','経度','
       lh','funiture_Score','animal_score','plant_score','behave_score','food_score','
       appliance_score','mic','clabel']
44
45
46 #デンドログラム表示関数
47 def show_den():
48
49     #codecsで読み込むと自動でunicodeにしてくれる
50     with codecs.open("fordend.csv", "r", "Shift-JIS", "ignore") as file:
51         df = pd.read_table(file,delimiter=",",names=names,encoding="Shift-JIS")
52
53         #format指定早い
54
55         df['時間'] = pd.to_datetime(df['時間'],format='%Y%m%d %H:%M:%S')
56         frame_time = pd.to_datetime(df['時間'],format='%Y%m%d %H:%M:%S')
57         df.set_index('時間', inplace=True)
58
59     #図クリア
60     plt.clf()
61
62     #ログから必要なデータをリスト化
63     last=df[len(df)-2:len(df)-1]
64     print(last)
65     word=df['clabel'].to_list()
66     loc_label=df['mic'].to_list()
67     LFHF=df['lh'].to_list()
68     list_time=frame_time.to_list()
69
70     print('データ数', len(df))
71
72     #カメラなし
73     #Z = linkage(df.drop("mic",axis=1).drop("clabel",axis=1).drop("funiture_Score
           ",axis=1).drop("animal_score",axis=1).drop("plant_score",axis=1).drop("
           behave_score",axis=1).drop("food_score",axis=1).drop("appliance_score",
           axis=1),method="ward",metric="euclidean")
74     #カメラあり
75     Z = linkage(df.drop("mic",axis=1).drop("clabel",axis=1),method="ward",metric="
           euclidean")
76     #最新のデータにラベル付け、label空白あるから穴埋め置換
77
78     #マイクラベルが存在するところに,その時のカメララベル[1]をくっつける
79     row1 = len(df)
80     for i in range(len(df)):
```

```
81
82          if pd.isnull(df['mic'][i])==True:
83              continue
84          df['mic'][i]=df['mic'][i]+df['clabel'][i]
85
86
87      df[row1-1:]=df[row1-1:].fillna("最新")
88      df_label=df['mic'].fillna(" ")
89
90      #leaf_rotation=0がラベルの向き、orientation="top"がクラスタリングの図の方向
91      #color_threshold=xでユークリッド平方距離がx以上を同色で表示
92      #above_threshold_color="color"でユークリッド平方距離がx以上を"color"色に染める
93      ##最新ラベルとラベルされているデータの距離が2000以内の場合同じ場所として判別#####
94      dendrogram(Z, labels=df_label,leaf_rotation='vertical',leaf_font_size=16,
            color_threshold=1700,above_threshold_color='gray')
95
96
97      #各データのクラスター番号出力
98      group = fcluster(Z, 1700, criterion='distance')
99      global camera11
100     ##################各クラスター番号がどのラベルなのか求める処理##############
101     loc_c = []
102     loc_d = []
103     loc_b = []
104     loc_num = []
105     stress_data= [[]]
106     H=[[]]
107     S=[]
108
109     for num in group:
110         loc_num.append(num)
111
112     loc_b = list(zip(loc_num, loc_label))
113     print(camera11)
114     print(group)
115
116     #ラベルはほとんどが'nan'なのでラベルがあったらそのクラスター番号を関連付け、
117     #(クラスター番号、場所)での配列を作る
118     for j,k in zip(loc_num, loc_label) :
119         if isinstance(k, str):
120             loc_d.append(j)
121             loc_c.append(k)
122     print(loc_d)
123     print(loc_c)
124     print("合体")
125     print(list(zip(loc_c,loc_d)))
126
127
128
129     ###############最新地の判断##################
130     nowpoint = group[length-1]
131     lastpoint = group[length-2]
132     now_time=list_time[length-1]
133
134     #最新地のクラスター番号を調べ、それに対応している場所を特定
135     if not nowpoint in loc_d:
136         location = '不明 不明'
137     else:
```

```
138             for s,t in zip(loc_d, loc_c):
139                 if nowpoint == s:
140                     location = t
141                     print('識別結果: '+t)
142
143         #マイクで場所と状況を一度に入れているので分割
144         loc_sp = location.split()
145         if len(loc_sp) == 1:
146             #入力ミスった場合
147             location = loc_sp[0]
148             situation= loc_sp[0]
149         else:
150             location=loc_sp[0]
151             situation=loc_sp[1]
152
153
154         ############場所が変わったかの判断と経過時間の算出
                ##############
155         #前回と今回の値が一緒だったら時間を加算、違ったら計測しなおす
156         #start_timeを用意しnow_timeと引き算して分を出す.行動変化したらそれをstaretimeに
                変更する
157         if lastpoint != nowpoint:
158             print("行動変化 " , nowpoint)
159             start_time = list_time[length−1]
160
161         else :
162             print("継続中 " , nowpoint)
163             for g in range(len(group)):
164                 if nowpoint != group[length −2−g]:
165                     start_time = list_time[length −1−g]
166                     break
167                 elif nowpoint==group[0]:
168                     start_time = list_time[0]
169
170         #行動経過時間の算出
171         total_time = now_time − start_time
172         total_second_time=int(total_time.total_seconds())
173         print('START ' , start_time)
174         print('NOW ' , now_time)
175         print("TOTAL " , total_time)
176
177         #ストレス計測 1
178         stress_data=math.floor(LFHF[−1] ∗ 10 ∗∗ 2) / (10 ∗∗ 2)
179
180          #検証用のグローバル変数宣言
181         global flag
182         global StarTime
183
184         #########コーピング :コーピング内容 ,図、状態############
185         #発動しない場合
186         #copen=コーピング発動したか(検証用)copeg=コーピングするか(検証用)cope0コー
                ピングするか(html用)
187         if total_second_time < 2700 and stress_data <5.0:
188             if stress_data <=2.0:
189                 state = '良好'
190                 figure='1.png'
191                 cope = 'No'
192             else :
193                 state = '注意'
194                 figure='2.jpg'
195                 cope = '警告 : 危機が迫っている'
```

```
196         copen=0
197         copeg=0
198         cope0='必要なし'
199
200     elif total_second_time >=2700 and stress_data <=2.0:
201         state = '注意'
202         figure='2.jpg'
203         cope = '警告：危機が迫っている'
204         copen=0
205         copeg=0
206         cope0='必要なし'
207
208     #発動する場合
209     else:
210         #スプレッドシートの単語から
211         #if ('computer' and 'monitor') in camera11:
212         #とりあえずPC操作にしておいた
213         # cope2 = 'コンピューター操作により'
214         # cope3 = '休憩に入り、目を休め'
215         #行動識別から
216         if 'パソコン' in situation:
217             cope2 = 'コンピューター操作により'
218             cope3 = '休憩に入り、目を休め'
219         elif '会議'in situation:
220             cope2 = '会議中なので'
221             cope3 = 'ちょっと休め、耐えろ'
222         elif '休'in situation:
223             cope2 = '休憩でも'
224             cope3 = '止めなさい'
225         elif '天井'in location:
226             cope2 = 'お前'
227             cope3 = '休みすぎ'
228         elif 'kichen'in location:
229             cope2 = '打ち合わせより'
230             cope3 = '止めた方がいいんじゃない'
231         else:
232             cope2 = 'その他の行動により'
233             cope3 = '気分転換しよう'
234
235         if ('person' and 'man') in camera11:
236             if last_jin[0]==0:
237                 cope2 = cope2 +'もしくは対人による可能性があるので'
238                 cope3 = cope3 + 'もしくは1人の時間を作る'
239
240         #とりあえす経過時間が長かった(90分以上,30分以上,それ以下)ら
241         if int(total_time.total_seconds()) > 1800:
242             print("長すぎ")
243             cope1 = '長時間行動より'
244
245         elif int(total_time.total_seconds()) > 2700:
246             print("長時間行動")
247             cope1 = '長時間行動と'
248         else:
249             print("短時間労働")
250             cope1 = '行動は短いが'
251             #cope3 = '行動を止めたほうがいい'
252
253         state = '要注意、コーピングを実行'
254         figure = '5.jpg'
255         copen = 1
256         cope= cope1+cope2+cope3
```

```
257         #ストレス検証 :3分間はコーピング実施もしくは無視
258         #発動されたら(だいたいLF/HFが5以上になった時)
259         #flagが０か１になる
260         #ランダムで０か１になる＝num,copeg
261
262         if flag==0:#ストレス検知&flag=0の時
263             flag=1
264             StarTime=datetime.now()
265             num= random.randint(0,1)
266             print(num)
267
268             if num==1:#コーピングする場合
269                 copeg=1
270                 cope0="する"
271
272             else:#コーピングしなきゃだけどしない場合
273                 copeg=0
274                 cope0="しない"
275
276         elif flag == 1:#ストレス検知&flag=1の時
277             if num==1:#コーピングする場合
278                 copeg=1
279                 cope0="する"
280
281             else:#コーピングしなきゃだけどしない場合
282                 copeg=0
283                 cope0="しない"
284
285     #検証用の処理､3分間はコーピングするかしないか確定させる
286     flag_time=datetime.now() − StarTime
287     print(flag_time)
288     if int(flag_time.total_seconds())>180:
289         flag = 0
290
291
292     ##########html制作##############
293     webopened = 'webページを作成します'
294     basestress = 5
295     print('stress: ',stress_data, ' ',state,' ',cope)
296     print(situation, webopened)
297
298     #htmlにグラフ画像はる
299     abc=[]
300     if len(LFHF)>20:
301         for t in list_time[−20:]:
302             t=t.strftime("%H:%M:%S")
303             abc.append(t)
304         defg=LFHF[−20:]
305     else:
306         for t in list_time:
307             t=t.strftime("%H:%M:%S")
308             abc.append(t)
309         defg=LFHF
310
311     xmin, xmax = 0, len(defg)
312     fig=plt.figure(figsize=(8,6))
313     plt.xticks(rotation=90)
314     plt.hlines(5,xmin, xmax, "red", linestyle="dashed")
315     plt.plot(abc,defg)
316     fig.savefig("graf.png")
317     plt.pause(3)
```

```
318     plt.close()
319
320     #コーピング発動html
321     with open('Image2.html', mode='w+', encoding="utf_8") as f:
322         data_line4=('<!DOCTYPE html>\n<head>\n<meta http−equiv="content-
                type" charset="utf-8">\n</head>\n<body>\n<div style="font-size:
                small">\n')
323         data_line5=('<br>\n</div>\n</body>\n<img src=' + figure + ' height="120
                "><img src=graf.png width="170"></html>')
324         data_lines=('場所 :' + location + '<br>\n状況 :' + situation + '<br><br>\n\n
                経過時間 :' + str(total_time) + '<br>\n')
325         data_lines2 = ('ストレス予測値 :' + str(stress_data) + ' / 状態 :'+ state+'<br>\
                n指令 :'+cope + '<br>\n' + cope0)
326         data_lines3=('Today :' + str(now_time) + '<br>\n')
327         f.write(data_line4)
328         f.write(data_lines3)
329         f.write(data_lines)
330         f.write(data_lines2)
331         f.write(data_line5)
332
333     #ストレスログ作成
334     with open('foo.csv', 'a',encoding="Shift-JIS") as f:
335         writer = csv.writer(f)
336         G = [now_time,datetime.now().strftime("%m.%d"),datetime.now().strftime("%H:%M
                :%S"),stress_data,copen,situation,copeg,basestress]
337         writer.writerow(G)
338
339     #最新ラベルの色付け,デンドログラム描画
340     #図の取得
341     #グラフ表示領域
342     #plt.ylim(0, 1500)
343     ax = plt.gca()
344     plt.tight_layout()
345     #plt.show()
346     xlbls = ax.get_xticklabels()
347     for lbl in xlbls:
348         if lbl.get_text()=="最新":
349             lbl.set_color("red")
350
351
352 ##################デンドログラム表示関数###############
353
354
355 #単語の6次元落とし関数
356 def sim(h,i,j,k):
357     try:
358         w2v[i][k]+=embed_map.similarity(h, add_pos_camera[i][j])
359     except KeyError:
360         error.append([add_pos_camera[i][j]])
361
362 #6軸の平均関数
363 def aveword(w,x,y):
364     w2v[w][x]=w2v[w][x]/y
365
366
367
368
369
370 #csvの有無の確認(なかったら作る)、ssからcsvへ同期のための行数取得
371 if os.path.exists("fordend.csv"):
372     with codecs.open("fordend.csv", "r", "Shift-JIS", "ignore") as file:
```

```
373             df = pd.read_table(file, delimiter=",",encoding="Shift-JIS")
374         #ラベル分(ないけど)引かれてるから+1
375         length=len(df)+1
376     else:
377         length=0
378     tmp_map=None
379     try:
380
381         flag=0
382         print("START")
383         StarTime=datetime.now()
384         while True:
385             data_all=[[]]
386
387             print("get data")
388
389             ile= pd.ExcelFile('fordend.xlsx')
390             ile2=pd.ExcelFile('fordend2.xlsx')
391             dr1 = ile.parse('Sheet1', index=False,header=None)
392             dr2 = ile2.parse('Sheet1', index=False,header=None)
393
394
395              #各ワークシートの値を配列に入れる
396             #各ワークシートの行(縦)の長さを取得、注意:行はxでも配列は0からだから最終行
                    はx−1
397
398             values1 = dr1
399
400             row1=len(values1)
401             #print(row1)
402             values2=dr2
403             row2=len(values2)
404             #print(values1)
405
406         #追加用の新規配列と処理
407             sabun=row1−length
408             print(sabun)
409         #lengthの更新
410             length=row1
411
412             sheet1=np.array(values1)
413             values2 = values2.values.tolist()
414
415             #最新のカメラテキスト取得
416             camera11=list(sheet1[row1−1,2])
417             #このままだた一文字ずつしかとらない
418             camera11 = ''.join(camera11)
419             camera11 = camera11.split(',')
420
421
422 #sabunがない場合(=0)、sabunがでるまで以下でスキップ
423             if sabun==0:
424                 print("no different data")
425             #ssに一つもデータが入ってなかったらデンドログラムも表示できないからスキッ
                    プ
426                 if row1==0:
427                 #time.sleep(120)
428                     continue
429                 #show_den()
430                 print("新規データを取得しています")
431
```

```
432             #ユーザーあたり100秒あたり100リクエストの制限
433                 time.sleep(30)
434                 continue
435
436             print(values2)
437
438         #[行,列]、「値1:値2」で値1~値2を取得(ただし値2は含まない)(値2は省略可、そのとき値q
                から配列の最後まで取得)
439         #−を指定で反対から(ex:−1は最後の値)
440             nowtime=list(sheet1[−sabun:,0])
441             nowtime = [s.replace('_', ' ') for s in nowtime]
442             sensa=list(sheet1[−sabun:,1])
443             camera=list(sheet1[−sabun:,2])
444
445         #リストの要素がカンマ区切りだから、splitする行ごとに[]で区切る
446             sensa=[i.split(",") for i in sensa]
447             camera=[i.split(",") for i in camera]
448
449
450 #ラベル同期
451         #pdのdetetimeの型に合わせるために変換
452             dfnowtime=pd.DataFrame({'time':[s.replace('_', ' ') for s in nowtime],'values_A':
                    list(range(len(nowtime)))})
453             dfnowtime['time'] = pd.to_datetime(dfnowtime['time'])
454             #zip(∗配列)で転置
455             dflabel=pd.DataFrame({'time':[s.replace('_', ' ') for s in list(zip(∗values2))[0]],'
                    value_B':list(zip(∗values2))[1]})
456             dflabel['time'] = pd.to_datetime(dflabel['time'])
457         #シート1と2の時間3sec以内でラベル同期
458             sync=pd.merge_asof(dfnowtime,dflabel,tolerance=pd.Timedelta('60s'),direction='
                    forward', on='time')
459
460
461         #cameraとposタグの対応
462             pos_camera=[np.array(nltk.pos_tag(i,tagset='universal'),dtype=object) for i in
                    camera]
463             add_pos_camera=camera
464
465
466             #sabunの分配列用意,カメラタグ配列の2つ目を入れていく(ラベルの詳細化のため)
467             adca = [0] ∗ (sabun)
468             for i in range(len(camera)):
469                 adca[i]=camera[i][1]
470
471
472         #word2vecのために進行形を原型に
473             for i in range(sabun):
474                 for j in range(len(pos_camera[i])):
475                     if pos_camera[i][j][1]=="VERB" and pos_camera[i][j][0].endswith('ing'):
476                         pos_camera[i][j][0]= lemmatizer.lemmatize(pos_camera[i][j][0], 'v')
477                 #cameraとposタグの合体
478                     add_pos_camera[i][j] = '_'.join(pos_camera[i][j])
479                 #add_pos_camera[i]=[inner[0]+'_'+inner[1] for outer in pos_camera[i] for
                        inner in outer]
480
481 #word2vec用関数とエラー処理
482             w2v = [[0] ∗ 6 for i in range(sabun)]
483             error=[]
484
485             for i in range(sabun):
486                 #家具、動物、植物、動物、行動、食品、家電
```

```
487                 for j in range(len(add_pos_camera[i])):
488                     sim("furniture_NOUN",i,j,0)
489                     sim("animal_NOUN",i,j,1)
490                     sim("plant_NOUN",i,j,2)
491                     sim("behave_VERB",i,j,3)
492                     sim("food_NOUN",i,j,4)
493                     sim("appliance_NOUN",i,j,5)
494 
495                 for j in range(6):
496                 #エラーは足していないから引く(/6はkeyerrorが6回起きるから)
497                     aveword(i,j,(len(add_pos_camera[i])−len(error)/6))
498                     error=[]
499 
500 
501 #nowtimeとsensa,w2v,miclabel,カメラタグの合体
502     #sumd1=np.column_stack((sync.value_B,adca))
503         add_data=np.concatenate((np.column_stack((nowtime,sensa)),np.column_stack((
                w2v,np.column_stack((sync.value_B,adca))))),axis=1)
504 
505     #add_data=np.concatenate((np.column_stack((nowtime,sensa)),np.column_stack((
            w2v,sync.value_B))),axis=1)
506     #csvに差分追加保存
507         with open('fordend.csv', 'a', newline='',encoding='Shift−JIS') as f:
508             writer = csv.writer(f)
509         #2次元配列をまとめて保存
510             writer.writerows(add_data)
511 
512         show_den()
513         time.sleep(30)
514 
515 
516 except KeyboardInterrupt:
517     #最後にkoo.csvと決定木を作成しておわり
518     stress_decision_tree.tree()
519     print('決定木を作成した')
520     print('!!FINISH!!')
```

A. 6　決定木分析するソースコード

決定木分析するソースコード A.6 を示す.

ソースコード A. 6: stress-decision-tree.py

```
1  from sklearn import tree
2  from sklearn.tree import DecisionTreeClassifier
3  from sklearn.tree import export_graphviz
4  import pydotplus
5  from IPython.display import Image
6  import numpy as np
7  import pandas as pd
8  import codecs
9  import csv
10 import os
11 import pandas as pd
12 import math
```

```
13
14 from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
15 #from matplotlib import pyplot as plt
16 #from matplotlib import rcParams
17
18 def tree():
19     S=[]
20     H=[]
21     names=['時間','温度','湿度','大気圧','輝度','動体検知','加速度x','加速度y','加速度z','角
           速度x','角速度y','角速度z','地磁気x','地磁気y','地磁気z','体温','心拍数','GSR','緯度
           ','経度','lh','funiture_Score','animal_score','plant_score','behave_score','food_score
           ','appliance_score','mic','clabel']
22
23     with codecs.open("fordend.csv", "r", "Shift-JIS", "ignore") as file:
24         df = pd.read_table(file,delimiter=",",names=names)
25
26
27     ond=df['温度'].to_list()
28     std=df['湿度'].to_list()
29     tka=df['大気圧'].to_list()
30     hkr=df['輝度'].to_list()
31     ksx=df['加速度x'].to_list()
32     ksy=df['加速度y'].to_list()
33     ksz=df['加速度z'].to_list()
34     kkx=df['角速度x'].to_list()
35     kky=df['角速度y'].to_list()
36     kkz=df['角速度z'].to_list()
37     tjx=df['地磁気x'].to_list()
38     tjy=df['地磁気y'].to_list()
39     tjz=df['地磁気z'].to_list()
40     jnk=df['動体検知'].to_list()
41     tio=df['体温'].to_list()
42     gsr=df['GSR'].to_list()
43     snp=df['心拍数'].to_list()
44     word=df['clabel'].to_list()
45     loc_label=df['mic'].to_list()
46     LFHF=df['lh'].to_list()
47
48
49
50     for i in LFHF:
51         if i > 5.0:
52             st_num='要注意'
53         elif i <= 2.0:
54             st_num='良好'
55         else :
56             st_num='注意'
57         S.append(st_num)
58     #生体センサ―もろもろの多次元配列制作
59     for a,b,c,d,e,f,g,h,i,k,l,m,n,o,p,q,r,s in zip(ond,std,tka,hkr,ksx,ksy,ksz,kkx,kky,kkz,tjx,
           tjy,tjz,jnk,tio,gsr,snp,S):
60         h = [a,b,c,d,e,f,g,h,i,k,l,m,n,o,p,q,r,s]
61         H.insert(len(H),h)
62     #koo.csvに書き込む。最後のデータがひたすら書き込まれるんだけどどうしよう…
63     #長さを比較して差分だけ書き込もう,イエア！
64     ds = pd.read_csv("koo.csv",encoding="SHIFT-JIS")
65     if len(H)>len(ds):
66             print(len(H),len(ds),H[1])
67             leng = len(H)+1−len(ds)
68             with open('koo.csv', 'a') as f:
69                 writer = csv.writer(f)
```

```
70                  for i in range(leng):
71                  #print(H[-i])
72                      writer.writerow(H[i])
73
74
75      #Graphvizのdot.exeがあるフォルダのパス
76      os.environ["PATH"] += os .pathsep + "C:/Users/wasaza/Desktop/研究/Anbient/
            IOT/bakkuappu/graphviz/release/bin"
77      #clf = svm.SVC(kernel='rbf', C=10, gamma=0.1)
78      with codecs.open("koo.csv", "r", "Shift-JIS", "ignore") as file:
79          df = pd.read_table(file,delimiter=",")
80      #print(df)
81      #df_tmp = df.drop(['Unnamed: 0'],axis=1)…csvによって書くか書かんか
82      df_tmp = df
83      df_tmp.head()
84
85      src = df_tmp.drop('Assessment',axis='columns')
86      #説明変数
87      trgt = df_tmp['Assessment']
88      #目的変数
89      #print(src)
90      #print(trgt)
91
92      #決定木つくる
93      clf = DecisionTreeClassifier(random_state = 0)
94      clf.fit(src,trgt)
95      DecisionTreeClassifier(random_state = 0)
96      print("決定木作成")
97      #図作る
98      dot_data = export_graphviz(clf,feature_names=src.columns,class_names=['注意','良好
            ','要注意'],filled=True,rounded=True)
99      graph = pydotplus.graph_from_dot_data(dot_data)
100     graph.write_png('tree.png')
101     Image(graph.create_png())
102     #print(graph)
```