2021 年度　卒業論文

AR を利用した熱中症予防システム

Heatstroke Prevention System Using Augmented Reality

指導教員　田中二郎　教授

早稲田大学 基幹理工学部 応用数理学科

インタラクティブプログラミング研究室

1W182274-0 花田 義樹

概要

日本では毎年夏になると、数多くの患者が熱中症で緊急搬送されている。若い世代を中心に運動中に熱中症の症状をうったえる患者も多い。本研究では、夏場にウォーキングやジョギングなどの軽い運動を行う際に使用することを想定し、熱中症の予防を試みる拡張現実（Augmented Reality, AR）を用いたシステムを開発した。スマートウォッチと咽喉マイクを用いて運動・休憩・水分補給の検出を行うことによってユーザーが晒されている熱中症のリスクをリアルタイムに算出し、これに基づいてエフェクトをヘッドマウントディスプレイ（Head Mounted Display, HMD）上に表示することによって可視化した。本システムを利用することによってユーザーは運動に集中しながらでも熱中症のリスクの上昇に気づくことができ、適切なタイミングや量で休憩や水分補給を行う手助けとし、夏場の運動の快適性と安全性の向上を図った。

謝辞

本研究をすすめるにあたって、指導教員の田中二郎教授には丁寧なご指導およびご助言をいただきました。研究室生活も北九州キャンパスでの生活も初めてであった私に様々な心構えや研究の進め方など非常に気にかけていただきました。また、インタラクティブプログラミング研究室の皆様にはわずか 1 年という短い間でしたが、研究におけるアドバイスや実装におけるご協力などを頂き大変お世話になりました。皆様から教わったことはこの先修士論文に向けた研究活動はもちろん、社会人生活としても役に立つと考えております。研究に協力してくださった皆様に心より御礼申し上げます。

目次

第 1 章

はじめに

1.1 はじめに

地球温暖化の進行に伴って、夏の暑さは厳しさを増している。日本では毎年夏になると、熱中症で救急搬送された人が何人いるといったようなニュースが毎日のように流れ、学校の部活動や市民ランナー等のスポーツの場に向けても積極的な熱中症対策が呼びかけられているが、それでも熱中症に倒れる人は後を絶たない。運動中に熱中症に倒れる人の多くは、適切な休憩や水分補給を行わなかったことによるものであるが、その原因として「まだ自分は休憩しなくても大丈夫」や「周りの人も大丈夫そうだから自分もまだ大丈夫だろう」といったように、自分の感覚や周囲の雰囲気で判断してしまったりすることによる場合がある[1]。

本研究では、ユーザーが運動を行いながらリアルタイムに今自分の体がどの程度熱中症のリスクに晒されているかを知ることができ、リスクが高まっていることに気づくことができれば、ユーザーが感覚や雰囲気で判断することなく適切に休憩・水分補給を行うことができるのではないかという仮説を立て、これを実現するためのシステムを提示する。

第 2 章

目的とアプローチ

2.1 背景

日本の夏場において、熱中症によって救急搬送された人の数は毎年約 40000 人～95000 人にものぼる[2]。運動中に熱中症にかかる患者も多く、学校の部活動や体育の授業等で夏場に運動を行う機会が多い 7-18 歳の世代では総患者数のうち 50%近くが運動中に熱中症にかかっている[3]。運動時の熱中症対策として、こまめな休憩や水分補給を行うよう意識することはもちろんであるが、適切なタイミングや量を運動しながら把握することは難しい。人間は体内の水分が 2%失われるとのどの渇きを感じ、運動能力の低下が始まる[4][5]。したがって、のどが渇いたと感じたときには既に熱中症の引き金となる脱水症状が始まっているということになるため、「のどが渇いたら水を飲む」というのは運動時の水分補給として不適切である。

熱中症対策を行うシステムとして、スマートフォンを用いた水分補給管理アプリのようなものも存在するものの、これはユーザーが自ら水分補給をしたことを入力する必要があり、運動時に水分補給や休憩するべきタイミングをユーザーに伝達するようなシステムではない[6]。

2.2 仮説と目的

このような背景を踏まえ、運動中に今自分の体がどの程度熱中症のリスクに晒されているかをリアルタイムに知ることができ、必要以上に意識を向けなくてもリスクの高まりに気がつくことができれば適切に休憩や水分補給を行うことができるのではないかという仮説を立てた。この仮説を検証するために、システムがユーザーの運動・休憩・水分補給を検出し、ユーザーが晒されているリスクをリアルタイムに計算し、これを伝達および警告を行うことができるシステムの提案および設計を本研究の目的とする。

2.3 アプローチ

2.2 で述べた目的を実現するため、本システムでは拡張現実（Augmented Reality, 以下 AR）を利用する。AR を利用することによりユーザーの視界に直接エフェクトをかけたり様々なデータを表示したりすることによって、ユーザーが直感的にリスクの上昇に気が付くことができるためである。さらに、スマートウォッチやスマートフォンのみを用いて同様の対策を行う場合、運動中にデバイスをこまめに見なければならないが、AR を用いればその必要がなく運動に集中することも可能となる。また、システムが運動・休憩・水分補給を検出するための手段としてスマートウォッチおよび咽喉マイクを利用する。

2.4 本システムが想定する利用者

本システムでは日本の夏場において、ウォーキングやジョギングなどの軽い運動を行う際に使用することを想定する。

第 3 章

関連研究

3.1 AR を用いた屋外ランニングサポートシステム [7]

この研究では AR を用いた屋外でランニングを行う際、①屋外で実際に走っているユーザーが使える機能、②屋内にいる人が屋外を走っている人が見ている景色を共有することができる機能の 2 つが実装されている。

①について、ユーザーはスマートグラスを装着することによって、景色を見ながら走行距離、運動時間、走行速度、消費カロリーといった情報をリアルタイムに確認しながらランニングを行うことができる。②について、屋内にいるユーザーは VR を用いることができるヘッドマウントディスプレイを装着して屋外ユーザーがランニングしながら見ている景色を見ることができる。この 2 つの機能を用いることによって離れた人ともランニングの情報を共有することができ、お互いにサポートすることができることで、ランニングという行為自体をより有意義なものにすることができる。

我々のシステムでは AR を用いてランニング中にリアルタイムに走行情報を確認できることの有効性について参考にしている。

3.2 リアルタイムな食品認識と食事摂取量検出に基づく食品記録システム [8]

この研究はいつどこで何を食べたかといった食品記録をより簡単に、より直感的にする方法を提供することを目的としたシステムある。

この研究では食品認識とカロリー推定をリアルタイムに行い、その結果をスマートグラスに表示すること、手首の動きと咀嚼音を記録し、食事を摂取したことを検出すること、以上の２つの機能を用いることによってユーザーは食事をするときにどのくらいのカロリーを摂取するかを気にしながら食べることができ、また食品記録をつけることが簡単になり後から振り返りやすくなることで、ユーザーの健康的な食生活を支援する。

我々のシステムでは水分補給の検出を行う際、本研究の食事を摂取したことを検出するシステムの部分を参考にしている。

3.3 AR による階段を利用することへの意欲向上システム [9]

この研究は階段を使うのが面倒だと思っている人や、階段を使おうと思っても継続できない人を対象として、AR を用いて階段を使うことへのモチベーションを高めることをねらったシステムである。

この研究ではユーザーは HMD を装着する。HMD に搭載されたカメラからの画像を解析して階段を検知し、階段の側面部分に標語を表示してユーザーのモチベーション向上を図っている。また、HMD に搭載された加速度センサーからの情報をもとにユーザーが階段を利用していることとその時間を計測することにより、階段を利用することで消費したと推定されるカロリーをリアルタイムに HMD 上に表示することによってさらなる意欲向上を狙った。

我々のシステムではリアルタイムに情報を HMD に表示することでユーザーの意識に与える効果について参考にしている。

3.4 温湿度センサーを用いた熱中症予防スマホアプリ[10]

スマートフォンを用いた熱中症予防システムの研究例である。この研究ではスマートフォンに温湿度センサーを接続し、現在位置の温度および湿度からユーザーが今いる環境の熱中症の危険度を推定、ユーザーに警告を行うことで、熱中症の予防を試みている。また、現在位置の危険度をサーバーに送信することで、多くのユーザーが使えば危険度が高い場所・低い場所が地図上で確認できるようになる。これによって危険度が低い場所を移動ルートとして選択できるようになったり、外出前に服装を選ぶ参考にするといった対策も行えるようになったり、さらなる熱中症予防効果が期待できるとしている。

3.5 スマートウォッチの体温センサーを用いた熱中症予防システム[11]

スマートウォッチの体温センサーを用いた熱中症予防システムの研究例である。熱中症の症状のひとつに体温の上昇も含まれている。この研究では、スマートウォッチに装備されている体温センサーを用いてユーザーの体温を継続的に測定し、深部体温を推定する。体温が一定の値を超えるとユーザーに通知を行い、水分補給や休憩を促している。これによりユーザーの熱中症のリスクが高まったら休憩を勧告できるようになり、効率の良い休憩や水分補給を行えるようになる。

本研究ではスマートウォッチの体温センサーは利用していないが、将来的にはこの研究を参考に体温の要素もシステムに組み込んでいきたいと考えている。

第 4 章

システムデザイン

4.1 システム全体の概要

本研究では、ユーザーにヘッドマウントディスプレイ、スマートウォッチ、スマートフォンに接続した咽喉マイク、以上の 3 つのデバイスを提供し、これら全てを装着しながら運動する。

図 4.1 に本システム全体の概略図を示す。本研究においてユーザーの体の状態を監視する手段として、運動もしくは休憩の継続時間測定と水分補給の検知の 2 つをシステムが行う。運動・休憩時間の測定はスマートウォッチの心拍数センサーを用いて心拍数データを取得し、一定の値以上あるいは以下が続いている時間を計測することで実現する。また、水分補給の検知はスマートウォッチのジャイロセンサーおよびローテーションセンサーのデータと咽喉マイクの音量データの 2 つを利用して総合的に判断する。以上のデータをサーバーを通じて HoloLens 2 に送信し、運動を継続している時間や前回水分補給してからの時間、水分補給の回数などのデータをまとめる。これを用いて、ユーザーが今どの程度熱中症のリスクに晒されているかを表す「ダメージ数」の増減を行い、ダメージ数が大きくなるとエフェクトを濃く、小さくなるとエフェクトを薄く表示することによって、ユーザーが意識していなくても運動を中断して休憩をとったり、休憩を終えて運動を再開したりする目安とする。

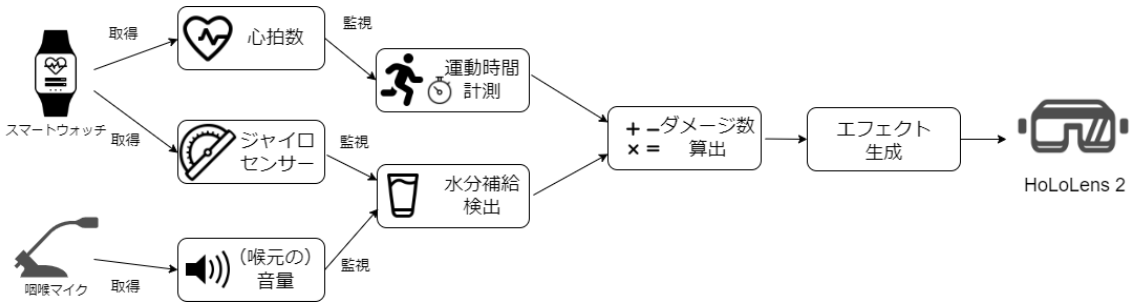


図 4.1 システムの概略図

図 4.2 に本システムの使用方法を示す。まず初めに、ユーザーが運動を開始すると、心拍数が上昇したことを検知して運動時間の計測、およびダメージ数の増加を開始する。運動時間が長くなったり長時間水分補給を行わなかったりして熱中症のリスクが高まってくると HoloLens 2 上のエフェクトを徐々に濃くしていき、ユーザーに休憩および水分補給を行うことを促す。ユーザーが休憩をとって心拍数が下がる、または水分補給を行うことにより熱中症のリスクが下がるとダメージ数を減少させ、エフェクトを徐々に薄くしていく。エフェクトが十分に薄くなった時をユーザーが運動を再開する目安とすることができる。この繰り返しによって、ユーザーは効率よくかつ安全に熱中症を予防しながら運動を行うことができる。

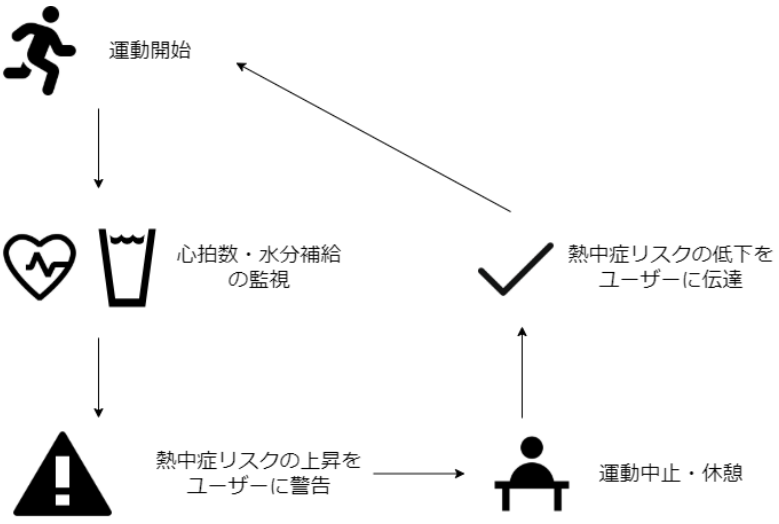


図 4.2 システムの使用方法

4.2 運動時間計測

運動を継続している時間の計測はスマートウォッチの心拍数センサーから取得する情報を利用する。

現在一般に市場に流通しているスマートウォッチには心拍数センサーが備わっているものがあるため、これを利用して心拍数の測定を行う。本研究では図 4.3 のようにスマートウォッチを左腕の内側に装着して利用する。

図 4.3 スマートウォッチの装着の例

本研究では心拍数が一定の閾値以上である間は運動を継続しているとみなし、この時間を計測する。反対に心拍数が閾値以下である間は運動をしていない休憩中の状態であるとみなし、休憩中の時間も計測する。これらの運動継続時間および休憩時間を用いてダメージ数の計算を行う材料とする。

閾値を決定するために、本システムが使用される状況下で想定される次の 3 つの条件のときの心拍数を計測した。

1.安静時

2.ウォーキング時

3.ランニング時

図 4.4 は安静時の心拍数を 10 分間計測したもの、図 4.5 はウォーキング時の心拍数を 10 分間計測したもの、図 4.6 はランニング時の心拍数を 10 分間計測した後、休憩時の心拍数を 5 分間計測したものである。これらより得られた結果は以下のとおりである。

1.安静時、心拍数はすべての時間で 90bpm 以下であった。

2.ウォーキング、ランニングともに運動中は常に 90bpm 以上であった。

3.ランニング後、3 分 30 秒程度で心拍数は 90bpm を下回った。

以上のデータより、閾値を 90bpm とし、これ以上であれば運動中と判断、これ未満であれば休憩中と判断するとした。

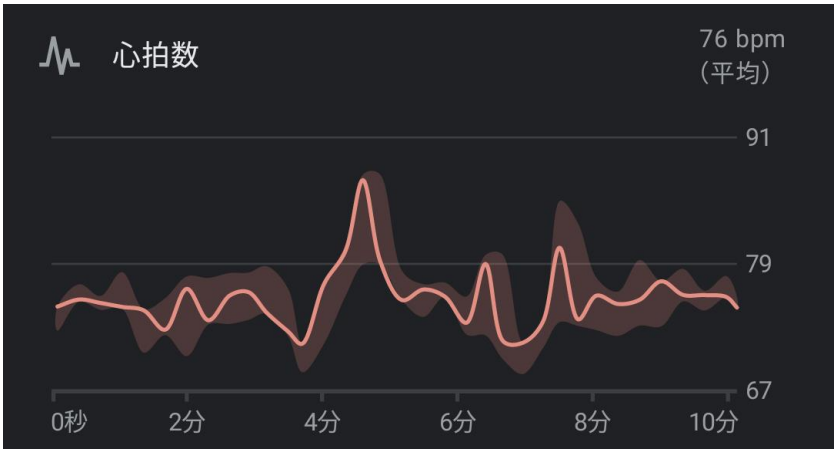


図 4.4 安静時の心拍数の推移

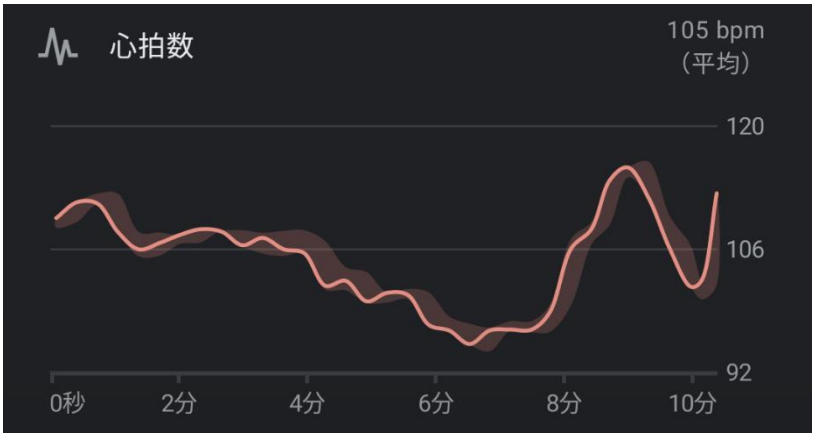


図 4.5 ウォーキング時の心拍数の推移

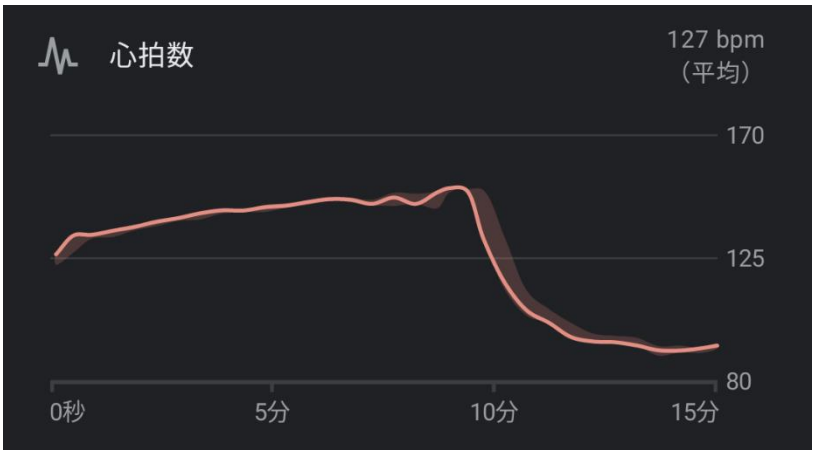


図 4.6 ランニング時とその後の休憩時の心拍数の推移

4.3 水分補給検知

水分補給の検知は次の 2 つがともに検知されたときに行う。

1.水筒やコップ等の容器を口元に運ぶときの手首の動き

2.水を飲み込む時の音（嚥下音）

4.3.1 容器を口元に運ぶときの手首の動きの検出

本研究では水分補給の際、水筒やコップ等の容器を口元に運ぶために手を上げるときの動きを検出し、かつウォーキングやランニング等の運動時に誤作動を起こさないようにするため、次の 4 つの条件における手首の動きの解析をスマートウォッチのジャイロセンサー及びローテーションセンサーを用いて行った。

1.水分補給をするときの手首の動き

2.ウォーキングをしているときの手首の動き

3.ランニングをしているときの手首の動き

4.休憩時、スマートフォンをいるときの手首の動き

まず、ジャイロセンサーからの情報について、次のような結論を得た。

スマートウォッチについているジャイロセンサーは、スマートウォッチ本体が移動したとき、その加速度を計測するものである。

図 4.7 は 1〜4 の各行動を行っているときのジャイロセンサーの x 軸方向成分の値の推移の例である。

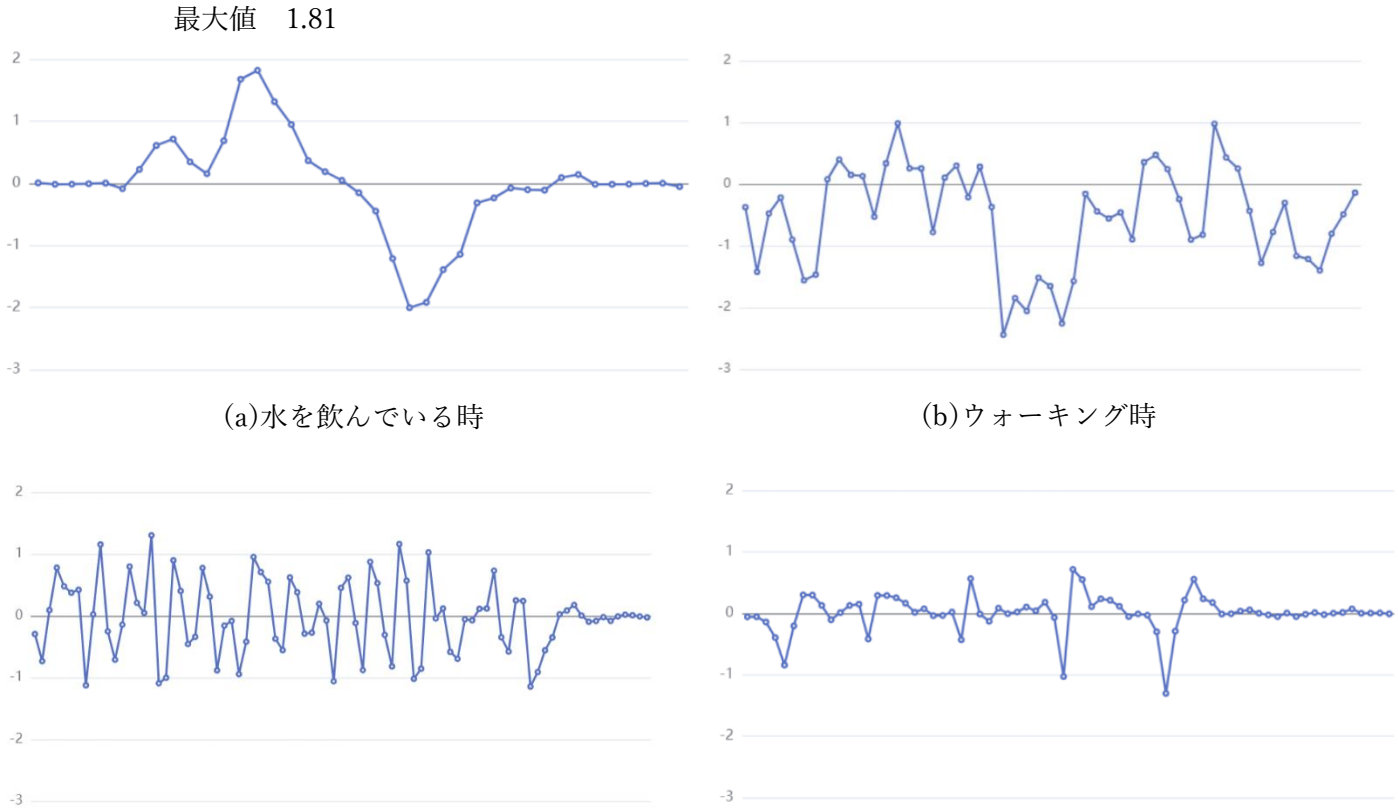
これらの図のように、次のような結論を得た。

1.水を飲む時、20 回中 18 回 1.80rad/s 以上になった。

2.ウォーキングをしている時、そのほとんどで 1.80rad/s 未満だった。

3.その他の行動を行ったとき、1.80rad/s を超えることはなかった。

これらより、ジャイロセンサーの x 軸方向成分が 1.80rad/s を超えたことを水を飲んだことを検知する条件の一つとした。



(a)水を飲んでいる時　(b)ウォーキング時

(c)ランニング時　(d)スマートフォン操作時

図 4.7 各行動時のジャイロセンサーx 成分の値の推移

また、ローテーションセンサーからの情報について、次のような結論を得た。

スマートウォッチのローテーションセンサーは、スマートウォッチ本体の傾きの回転ベクトルを計測するセンサーである。

図 4.8 は 1〜4 の各行動を行っている時のローテーションセンサーの y 軸方向成分の値の推移の例である。

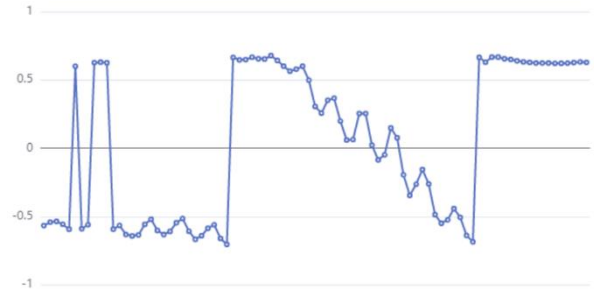
これらの図のように、次のような結論を得た。

1.水を飲むとき、20 回中すべてでローテーションセンサーの y 軸方向成分が-0.3 以下になった。
2.ランニングをしている時も y 軸方向成分が-0.3 以下になることがあったが、水を飲むときに比べその頻度は低かった。
3.その他のアクテビティを行ったとき、ローテーションセンサーの y 軸方向成分が-0.5 以上-0.3 以下の範囲に入ることはなかった。

これらより、ローテーションセンサーの y 軸方向成分が-0.3 以下であることを水を飲んだことを検知する条件の一つとした。
ジャイロセンサーとローテーションの 2 条件を手首の動きの条件として課すことにより、水を飲むときにのみシステムが作動するようにできる。

(a)水を飲んでいる時

(b)ウォーキング時



(c)ランニング時

(d)スマートフォン操作時

図 4.8 各行動時のローテーションセンサーy 軸方向成分の値の推移の例

4.3.2 水を飲みこむときの嚥下音の検出

本研究では水分補給の際、口に含んだ水を飲みこむときの音を検出するために、図4.9 のように咽喉マイクを喉元に装着して使用する。これにより、水を飲みこんだ時のわずかな喉の振動も検知することができる。

図 4.9 咽喉マイクを装着した例

水を飲むと、図 4.10 のようにマイクからの音量が水を飲みこんだ瞬間だけ増大する。これを用いて水を飲みこんだ時の嚥下音の検出を行う。

20 回実験を行った結果、そのすべてでマイクからの音量の値が 6000 を超えたため、これを水を飲みこんだと判断する基準とする。

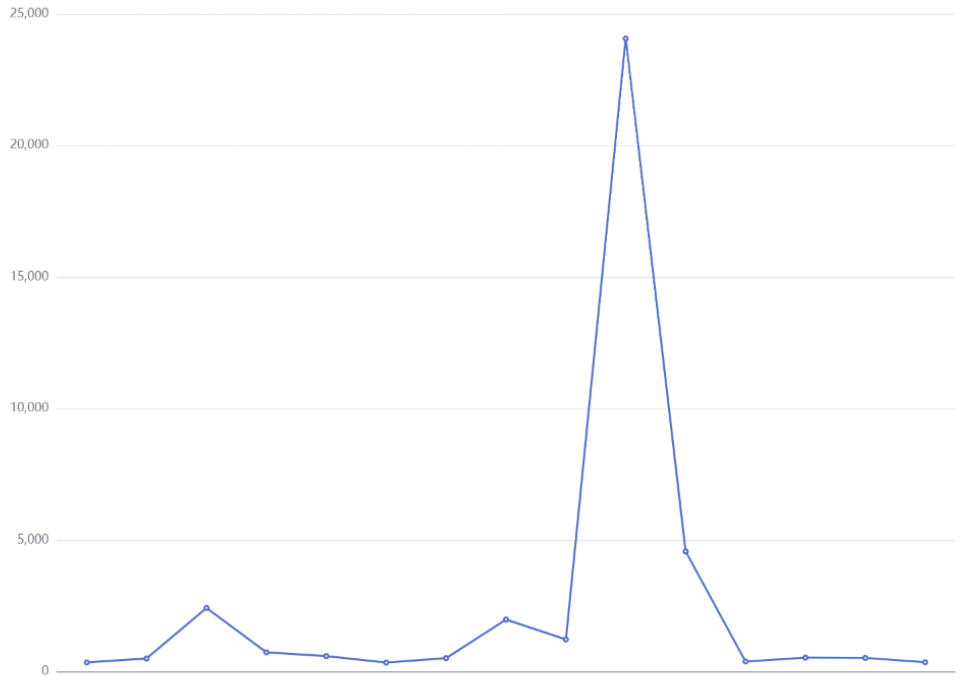


図 4.10 水を飲みこんだ時の咽喉マイクからの音量の値の推移

4.3.3 水分補給検知の条件

以上より、水分補給を行ったことを検知する 3 つの条件を次のように定める。

①ジャイロセンサーの x 軸方向成分が 1.80 rad/s 以上になること

②①から 2 秒以内に加速度センサーの y 軸方向成分が-0.3 以下になること

③①から 10 秒以内に咽喉マイクからの音量が 6000 以上になること

以上の 3 つの条件がすべて揃ったときにシステムが水分補給を行ったと判断する。

4.4 ダメージ数

「今自分の体がどの程度熱中症のリスクに晒されているか」を示す値として「ダメージ数」という概念を導入する。ダメージ数は運動をしている間徐々に増えていき、休憩をしている間は徐々に減っていき、また水分補給をすることでも減らすことができる。これにより、運動中でも熱中症のリスクにリアルタイムかつ簡単に気づくことができる。

熱中症対策として、こまめに水分補給を行い 30 分に 1 回程度休憩を行うことが望ましい[3][12]とされているため、ダメージ数の増減を以下のように設定する。

・ダメージ数は 0 から 100 までの範囲内とする
・運動を継続している間、1 分につき+5　（12 秒で+1）
・休憩をとっている間、1 分につき-10　（6 秒で-1）
・水分補給を行うと-10　ただし 1 度水分補給をしてから 5 分間は減算しない
・ダメージ数が 75 を超えるとそろそろ休憩を取るようユーザーに促す
・ダメージ数が 90 を超えるとすぐに休憩を取るようユーザーに警告を行う

これにより、5 分おきに水分を取りながら運動して 30 分運動を続けるとダメージ数が 90 を超え、ユーザーに休憩を取るよう促すことができる。

休憩中のダメージ量の減少量は心拍数が下がるまでのタイムラグを考慮して多めに設定している。

一度に多量の水を摂取することによってダメージ数が急激に減ってしまうことを防ぐため、1 度水分補給をしてから 5 分間は減算しないというルールを設けている。

また、ダメージ数に応じてエフェクトを発生させることで、直感的にかつ意識していなくても気づくことができる警告を発する。

第 5 章

システムの実装

5.1 開発環境

本研究におけるシステム開発で使用したアプリケーションは以下の通りである。

5.1.1 Unity 3D

Unity[13]は 2D および 3D のゲーム、アプリ、エクスペリエンスを作成できるツールである。現在、世界中で OS を問わず様々なゲームやアプリが Unity を用いて開発されているほか、AR を用いたアプリケーション開発も可能である。ユーザーは C#を用いたプログラミングが可能であり、本研究でも C#を用いて開発を行った。本研究では Unity 2020.3.16f1 を用いて開発を行った。

5.1.2 Mixed Reality Toolkit

Mixed Reality Toolkit (MRTK)[14]は Microsoft が提供する、Unity 上で利用可能な空間操作および UI 用のクロスプラットフォーム入力システムと構成要素を提供するライブラリである。

5.1.3 Android Studio

Android Studio[15]は Google が提供する、Android アプリ開発用の統合開発環境である。ユーザーは Java および Kotlin による開発が可能であり、本研究では主に Java を用いて、スマートフォンとスマートウォッチ向けのアプリケーションを開発する際に使用し、Android Studio2020.3.1 を用いて開発を行った。

5.2 ハードウェア

本研究を行うにあたって、必要とするデバイスとデバイスに必要な機能は以下の 4 つである。

1.文字やエフェクトの生成と投影、インターネットへの接続が可能な HMD
2.心拍数の計測とインターネットへの接続が可能なスマートウォッチ
3.喉元の振動を音に変換することが可能な咽喉マイク
4.3 の咽喉マイクと接続ができ、インターネットへの接続が可能なスマートフォン

これらを踏まえ、HMD として Microsoft HoloLens 2、スマートウォッチとして Fossil Smartwatch，スマートフォンとして Google Pixel 4a (5G)を選定した。

5.1.1 HMD

図 5.1 Microsoft HoloLens 2

図 5.1 に示したような Microsoft HoloLens 2 は Unity によるアプリケーション制作が可能であり、文字やエフェクトの生成と投影、インターネットへの接続が可能である。これらの機能を利用し、スマートウォッチおよびスマートフォンからのデータをインターネット経由で受信し、HoloLens 2 上でデータを処理してダメージ数の計算、エフェクトの生成を実現することができる。

5.2.2 スマートウォッチ

図 5.2 Fossil Smartwatch

Fossil Smartwatch は心拍数および本体の向きを計測することが可能なスマートウォッチであり、インターネットへの接続も可能である。Google が提供する Wear OS を搭載していることから、Android Studio によるアプリケーション開発が可能であり、アプリケーション開発が容易であるという点で本研究で採用した。このスマートウォッチではなくても、Wear OS を搭載しており心拍数センサーとジャイロセンサー、ローテーションセンサーがついているスマートウォッチであれば他の機種でも本システムを利用することができる。

5.2.3 咽喉マイクおよびスマートフォン

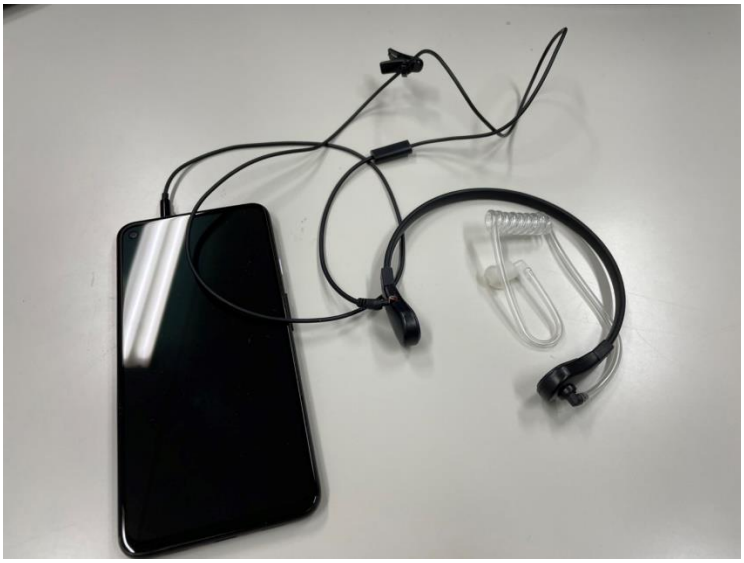


図 5.3 咽喉マイクに接続した Pixel 4a (5G)

本研究では一般に市場に流通している咽喉マイクを Pixel 4a(5G)に接続したものを採用した。本研究では咽喉マイクの機種に特に指定はない。スマートフォンの機種にも特に指定はないが、Android Studio による開発が容易であることから Pixel 4a(5G)を採用した。Android を搭載していて咽喉マイクと接続可能なスマートフォンであれば他の機種でも本システムを利用することができる。

5.3 スマートウォッチのセンサーデータの取得

本章部分の開発は Android Studio を利用し、プログラミング言語は Java を利用している。

既に 4.2 および 4.3、5.2.2 で述べたとおり、本システムでは Wear OS に搭載された心拍数センサー、ジャイロセンサー、ローテーションセンサーのデータを取得して利用する。

これらのデータを取得するために、Android アプリケーション開発者が利用できる SensorManager を使用し、以下の 3 つのセンサーのデータを取得した。[16]

表 5.1 利用したセンサーの一覧

センサー	センサーの内容
TYPE_HEART_RATE	心拍数を取得する
TYPE_GYROSCOPE	ジャイロスコープの値を取得する
TYPE_ROTATION_VECTOR	ローテーションベクトルセンサーの値を取得する

これらのセンサーを利用するため

・android.hardware.Sensor

・android.hardware.SensorEvent

・android.hardware.SensorEventListener

・android.hardware.SensorManager

の 4 つをインポートした。

次に、表 5.1 の 3 つのセンサーを用いるため SensorManager を取得し、また 3 つのセンサーを Sensor として定義した。これらを行っているソースコードをソースコード 1 に示す。

ソースコード 1　SensorManager の取得および Sensor の定義部分

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
Sensor heartSensor = sensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
Sensor gryoSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
Sensor rotationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);
```

次に、センサーの値が変化したときにサーバーに送信するデータを変更できるようイベントリスナーをソースコード 2 のように登録した。ソースコード 2 は心拍数センサーの登録の部分であるが、ジャイロセンサー、ローテーションセンサーに関しても同様である。

ソースコード 2　イベントリスナーの登録（心拍数センサー）

```
sensorManager.registerListener(new SensorEventListener(){

    @Override
    public void onSensorChanged(SensorEvent event) {
        BPM = String.format("%.1f", event.values[0]);
        mTextView.setText("BPM: "+BPM+"¥n "+ "Gro: ("+Gyo[0]+","+Gyo[1]+","+Gyo[2]+")¥n "+ "Rotation:
("+Rotation[0]+","+Rotation[1]+","+Rotation[2]+","+")");
        ServerSample.heartRate = Float.valueOf(BPM);
        ServerSample.send = true;
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {

    }
},heartSensor, SensorManager.SENSOR_DELAY_NORMAL);
```

BPM は心拍数を格納する string 型変数、Gyo と Rotation はそれぞれジャイロセンサーとローテーションセンサーの(x,y,z)の 3 成分の値を格納する string 配列型変数である。システムが動作していることを分かりやすくするためスマートウォッチ上に現在のセンサーの値を表示させるために TextView を用いている。

本システムではサーバーに送信する際、ServerSample.java という別のスクリプトを用いており、この部分の説明は 5.5 章で行っている。

このようにして制作したスマートウォッチのアプリケーションが動作している様子を図 5.4 に示す。現在の心拍数、ジャイロセンサー、ローテーションセンサーの値が取得できていることが分かる。

図 5.4 制作したスマートウォッチアプリの動作の様子

5.4 咽喉マイクの音量の取得

本章部分の開発は Android Studio を利用し、プログラミング言語は Java を利用している。

既に 4.3 および 5.2.3 で述べた通り、嚥下音の検出をスマートフォンに接続した咽喉マイクからの音量を取得することにより行う。咽喉マイクとスマートフォンは一般的なイヤホンジャックを用いて接続する。

音量を検出する部分のソースコードをソースコード 3 に示す。

ソースコード 3　咽喉マイクによる音量検出

```
AudioRecord soundDetection = new AudioRecord(MediaRecorder.AudioSource.MIC, 8000,
AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT, 20000);
soundDetection.startRecording();
ByteBuffer buff = ByteBuffer.allocateDirect(20000);
while(true)
{
    int r = soundDetection.read(buff, 20000);
    int mShortArrayLenght = r/2;
    byte[] bytes_pkg = buff.array();
    short[] short_buffer = byteArray2ShortArray(bytes_pkg, mShortArrayLenght);
    int max = 0;
    if (r > 0) {
        for (int i = 0; i < mShortArrayLenght; i++) {
            if (Math.abs(short_buffer[i]) > max) {
                max = Math.abs(short_buffer[i]);
            }
        }
    }
    ServerSample.volume = max;
    ServerSample.send = true;
    System.out.println("max ================= "+max);
}
```

このソースコードを利用することにより、咽喉マイクからの音量をリアルタイムにサーバーに送信することが可能となる。こちらのシステムでも 5.3 と同様に本システムではサーバーに送信する際、ServerSample.java という別のスクリプトを用いており、この部分の説明は 5.5 章で行っている。

このようにしてスマートフォンから取得したデータを Android Studio 上に表示させている様子を図 5.5 に示す。咽喉マイクからの音量が取得できていることが分かる。

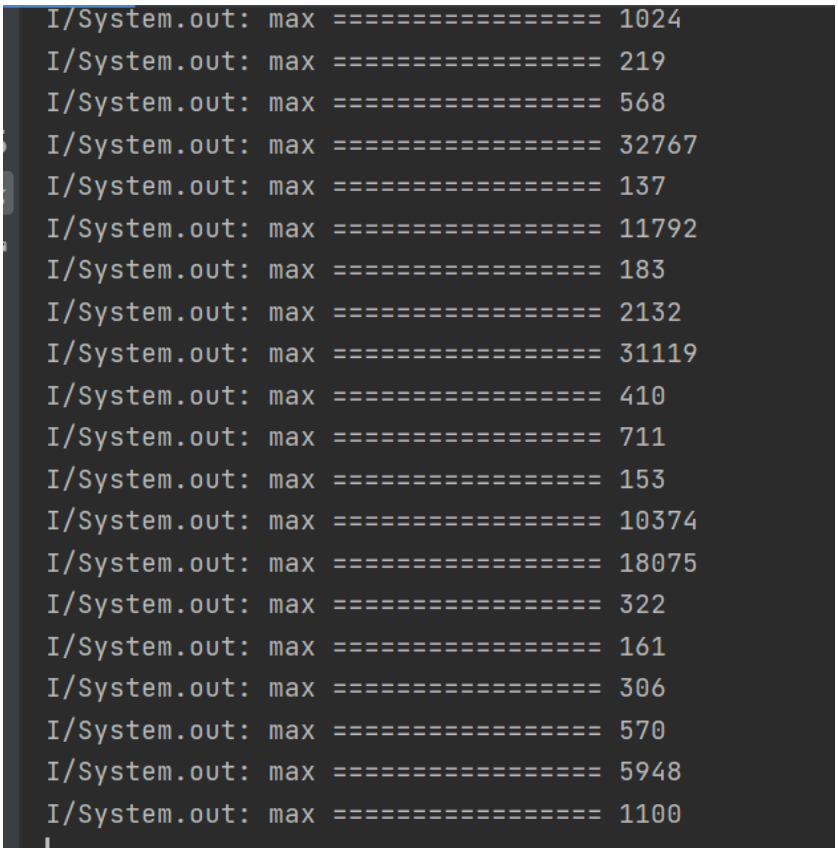


図 5.5 咽喉マイクからの音量を取得している様子の例

5.5 データの送受信

本章部分の開発はデータの送信部分は Android Studio を用いて Java で記述、受信部分は Unity を用いて C#で記述している。

スマートウォッチおよびスマートフォンからのデータの送信および受信には ServerSocket を利用して行っている。

スマートウォッチのデータを送信する部分のソースコードをソースコード 4 に示す。スマートフォンに関しても同様にして行っている。

ソースコード 4　データの送信を行うソースコード（スマートウォッチ）

```
public void run()
{
    System.out.println("==================Server started================");

    while(true){
        try{
            sc = serverSocket.accept();
            br = new BufferedReader(new InputStreamReader(sc.getInputStream()));
            pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(sc.getOutputStream())));
            while (true)
            {
                if(send)
                {

pw.println("BPM"+heartRate+"Gro:(X"+gyox+"Y"+gyoy+"Z"+gyoz+")Rot:(x"+rotx+"y"+roty+"z"+rotz+")");
                    String test =
"BPM"+heartRate+"Gro:(X"+gyox+"Y"+gyoy+"Z"+gyoz+")Rot:(x"+rotx+"y"+roty+"z"+rotz+")";
                    pw.flush();
                    send = false;
                }
            }
        }
        catch(Exception e){
            try{
                br.close();
                pw.close();
                sc.close();
                break;
            }
            catch(Exception ex){
                ex.printStackTrace();
            }

        }
    }
}
```

このソースコードではサーバーへデータの送信を開始するとスマートウォッチ上に「Server Start!」と表示させてから、心拍数、ジャイロセンサー、ローテーションセンサーの値を String 型の文字列形式でサーバーへの送信を行っている。

このようにして送信したデータを、Unity で受信するためのソースコードをソースコード 5 に示す。

ソースコード 5　データの受信を行うソースコード（スマートウォッチ）

```
void Start()
    {
        try
        {
            IPEndPoint ipep = new IPEndPoint(IPAddress.Parse(ipaddr), port);
            socket = new Socket(ipep.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
            socket.Connect(ipep);
            thread = new Thread(new ThreadStart(doWork));
            thread.Start();
        }
        catch(Exception e)
        {
            Debug.Log(e.Message);
        }


    }

    public void doWork()
    {
        Debug.Log("=========Watch Connectted========" );
        while (true)
        {
            byte[] buffer = new byte[1024];
            NetworkStream stream = new NetworkStream(socket);
            StreamReader reader = new StreamReader(stream);
           // message = reader.ReadLine();
            socket.Receive(buffer, SocketFlags.None);
            message = Encoding.ASCII.GetString(buffer);
            Debug.Log(message);
            process(message);

        }

    }
```

IP アドレスは Unity 内で設定を行う。このようにしてデータを受信し、Unity 上に表示させている様子を図 5.6 に示す。これらのデータを利用し、Unity 上で運動時間の計測、水分補給の検知、ダメージ数の計算、エフェクトの表示を行う。

ソースコード 4 およびソースコード 5 を用いて、スマートウォッチおよびスマートフォンから送信したデータを Unity 上で受信している様子を図 5.6 に示す。この図は 5.3 および 5.4 で制作したスマートウォッチとスマートフォンのアプリを起動し、2 つのデバイスと Unity を起動している PC を同じネットワークに接続した状態で、実際にデータの送信を行って撮影している。スマートフォンから心拍数、ジャイロセンサー、ローテーションセンサーの値、スマートフォンから咽喉マイクの音量の値を受信できていることが分かる。

図 5.6 Unity 上でデータを受信している様子の例

スマートウォッチから送られてきたデータはデータ 1 のように心拍数、ジャイロセンサー、ローテーションセンサーの値が列記された String 型の文字列の形で送信している。

データ 1　スマートウォッチから送られるデータの例

```
BPM:107.9Gro:(X1.80Y-3.31Z-2.64)Rot:(x0.0y-0.4z0.0)
```

そのため、これらを double 型、float 型の数値として利用できるようにするために、

message.substring を用いて使用したい部分だけを切り抜いた後、double.Parse および float.Parse を用いて double 型、float 型に変換する。この部分のソースコードをソースコード 6 に示す。今回はスマートウォッチからのデータで使用する値は心拍数、ジャイロセンサーx 成分、ローテーションセンサーy 成分のみなのでこの 3 つの値を取り出している。スマートフォンからのデータについても同様にして抽出を行っている。

ソースコード 6　スマートウォッチのデータを数値に変換する

```
string gyox = "";
        string roty = "";
        string BPMs = "";
        try
        {
            BPMs = message.Substring(message.IndexOf("M") + 1, message.IndexOf("G") - message.IndexOf("M") - 1);
            gyox = message.Substring(message.IndexOf("X") + 1, message.IndexOf("Y") - message.IndexOf("X") - 1);
            roty = message.Substring(message.IndexOf("y") + 1, message.IndexOf("z") - message.IndexOf("y") - 1);
            double test1 = double.Parse(gyox);
            double test2 = double.Parse(roty);
            BPM = float.Parse(BPMs);
```

これにより、運動時間の計測、水分補給の検知に必要なデータを全て Unity 上に集約させることができる。

5.6 運動時間の計測

5.5 でスマートウォッチから受信した心拍数センサーの値を用いる。

運動時間の計測を行う部分のソースコードをソースコード 7 に示す。

ソースコード 7　運動時間の計測

```
    private float ExerciseTime = 0;
    private float RestTime = 0;

void Update()
    {

        if(ConnectController.BPM > 90.0f)
        {
            ExerciseTime += Time.deltaTime;
        }

        if (ConnectController.BPM < 90.0f)
        { RestTime += Time.deltaTime; }
```

このソースコードでは心拍数が 90 以上のときは ExerciseTime に、心拍数が 90 未満の時は RestTime に経過時間を次々加算していくことにより、運動時間および休憩時間の計測を行っている。5.8 のダメージ数のシステムを製作する際、この ExerciseTime および RestTime の時間をダメージ数の計算要素に加えている。

5.7 水分補給の検知

5.5 でスマートウォッチおよびスマートフォンから受信したデータを利用する。

スマートウォッチのジャイロセンサーとローテーションセンサーの値から水分補給をするときの手首の動きを検知する部分のソースコードをソースコード 8 に示す。

このソースコードで、4.3.3 で述べた水分補給の検知を行う 3 条件のうち

①ジャイロセンサーの x 軸方向成分が 1.80 rad/s 以上になること

②1 から 2 秒以内に加速度センサーの y 軸方向成分が-0.3 以下になること

の 2 条件の判断を行っている。

ソースコード８　スマートウォッチのデータによる水分補給の検知

```
    public static DateTime timer = new DateTime(0);
    public static DateTime timer2 = new DateTime(0);
    public static DateTime judge = new DateTime(2001, 1, 1, 1, 0, 0, 0);
    public static TimeSpan limit1 = new TimeSpan(0,0,2);
    public static TimeSpan limit2 = new TimeSpan(0,0,10);

if (test1 > 1.80)
            {
                cond1 = true;
                timer = DateTime.UtcNow;

            }

            if (timer> judge && DateTime.UtcNow - timer < limit1)
            {
                cond1 = true;
                if (test2 < -0.30)
                {
                    cond2 = true;
                    timer2 = DateTime.UtcNow;
                }
                else
                {
                    cond2 = false;
                }
            }
            else
            {
                cond1 = false;
                if (test2 < -0.30)
                {
                    cond2 = true;
                }
                else
                {
                    cond2 = false;
                }

                if (timer2 < judge || DateTime.UtcNow-timer2 >
ConnectControlerPhone.limit2)
                {
                    timer = new DateTime(0);
                    timer2 = new DateTime(0);
                }
            }


            if (cond1 == true)
            {
                Debug.Log("Con1 true");
            }
            if (cond2 == true)
```

```
            {
                Debug.Log("Con2 true");
            }
            Debug.Log(timer);

        }
```

ソースコード 6 内で定義を行っているが、test1 はジャイロセンサーの x 成分、test2 はローテーションセンサーの y 成分の値を格納している。

条件①（cond1）が true になったときにタイマーをスタートさせ、２秒以内に条件②が true になった場合は cond1 と cond2 がともに true になるように、２秒以内に②が true にならなかった場合は cond 1 を false にすることで、条件②の時間制限を実現している。

続いて、スマートフォンからの咽喉マイクの音量データから嚥下音を検知するソースコードをソースコード 9 に示す。

ソースコード 9　咽喉マイクの音量による嚥下音検出

```
if (ConnectController.timer>ConnectController.judge && DateTime.UtcNow -
ConnectController.timer < limit2)
        {
            ConnectController.cond1 = true;
            ConnectController.cond2 = true;
            if (test3 > 6000)
            {
                cond3 = true;
            }
            else
            {
                cond3 = false;
            }
        }
        else
        {
            ConnectController.cond1 = false;
            ConnectController.cond2 = false;
            if (test3 > 6000)
            {
                cond3 = true;
            }
            else
            {
                cond3 = false;
            }
            ConnectController.timer = new DateTime(0);
            ConnectController.timer2 = new DateTime(0);
```

```
        }
```

test3 は咽喉マイクの音量を格納している。このソースコードでは、4.3.3 で述べた条件の 1 つである

③①から 10 秒以内に咽喉マイクからの音量が 6000 以上になること

の判断を行っている。ソースコード 8 と同様に、条件①から 10 秒以内に③が true になった場合は cond1,cond2,cond3 をすべて true に、10 秒以内に③が true にならなかった場合は cond1, cond2 を false にすることで、条件③の時間制限を実現している。

最後に、3 条件がすべて揃ったときに水分補給を行ったと判断を行う部分のソースコードをソースコード１０に示す。

ソースコード１０　水分補給を判断するプログラム

```
public class DrinkDetection : MonoBehaviour
{
    public TextMesh threeText;
    public static int dd = 0;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if(ConnectController.cond1 == true && ConnectController.cond2 == true  &&
ConnectControlerPhone.cond3 == true)
        {
            dd = 1;
            threeText.text = "Drink Detected!";
            ConnectController.timer = new DateTime(0);
            ConnectController.timer2 = new DateTime(0);
        }
        }else
        {
            dd = 0;
            threeText.text = "";
        }
    }
}
```

ソースコード 10 では、ソースコード 8 およびソースコード 9 で cond1,cond2,cond3 がすべて true になったときにのみ int 型変数 dd を 1 にすることにより、dd が 1 のときは水分補給を行ったと判断、dd が 0 のときは水分補給を行っていないと判断する。また、水分補給を検知を行ったと判断されたとき、「Drink Detected!」を表示する。

以上のプログラムによりプログラムが水分補給を検知している様子を図 5.7 に示す。ユーザーが各条件を満たしていない状態では上の画像のように「Stand-by」と表示されている。この状態から水分補給を行いこれをシステムが検知すると、下の画像のように Condition 1, Condition 2, Condition 3 がすべて Satisfied!(満たされている)と表示され、水分補給を検知したことを表す「Drink Detected!」が表示される。

図 5.7 システムが水分補給を検知している様子

（上）スタンバイ時　（下）検知時

5.8 ダメージ数とエフェクト

5.6 で計測した運動時間、5.7 で検出した水分補給を利用する。4.4 で記述したルールに従ってプログラムを行った。

まず、「水分補給を行うとダメージ数を-10 するが、1 度水分補給をしてから 5 分間は減算しない」というルールを実現するため、最後に水分補給を行ってからの時間を計測するソースコードを製作した。これをソースコード 11 に示す。

ソースコード 11 水分補給を行ってからの時間を計測するソースコード

```
if (DrinkTime == 0)
{
    if (DrinkDetection.dd == 1)
    {
        DrinkCounter = DrinkCounter + 1;
        DrinkTime += Time.deltaTime;
    }
}
else if (DrinkTime>0 && DrinkTime < 300.0f)
{
    DrinkTime += Time.deltaTime;
}
else if (DrinkTime > 300.0f)
{
    DrinkTime = 0;
}
```

このソースコードでは、DrinkTime が 0 のときにソースコード 10 で定義した dd が 1 になったら水分補給の回数を数える DrinkCounter を 1 増やし時間の計測を開始、DrinkTime が 300 になったら 0 にリセットを行う。0 より大きく 300 未満の時は DrinkCounter の値を変化させない。これによって、最後に水分補給してから 5 分間はダメージ数を変化させないというルールを実現している。

次に、運動時間、休憩時間、水分補給の回数の 3 つの要素を用いて実際にダメージ数を計算するソースコードをソースコード 12 に示す。

ソースコード 12 ダメージ数を計算するソースコード

```
        Damage = ((int)ExerciseTime/12) - ((int)RestTime / 60)-((DrinkCounter)*10);
        if (Damage > 100) {
            DamageN = 100;
        }
        else if(Damage < 0)
        {
            DamageN = 0;
        }
        else
        {
            DamageN = Damage;
        }
```

このソースコードでは Damage で実際の値を計算、DamageN で 0 から 100 までの値に修正している。これはユーザーに見える値は分かりやすくするためであり、システム内部では 0 以下の値や 100 以上の値になるよう設定している。

続いて、ダメージ数が大きくなるにつれてエフェクトを表示させる。本システムではダメージ数が大きくなるとユーザーの視界を赤く染め上げるようにした。このエフェクトを実現するためのソースコードをソースコード 13 に示す。

ソースコード 13　エフェクトを表示させるソースコード

```
private float alpha = 0.0f;
    private float damagef;
    private Image imgstatus;

    // Start is called before the first frame update
    void Start()
    {
        imgstatus = GameObject.Find("Panel").GetComponent<Image>();
    }

    // Update is called once per frame
    void Update()
    {
        damagef = DamageCounter.DamageN;
        //Debug.Log(damagef);
        alpha = damagef / 200.0f;
        //Debug.Log(alpha);
        imgstatus.color = new Color(1.0f, 0.0f, 0.0f, alpha);
    }
```

このソースコードでは、ダメージ数が大きくなると赤色の Canvas の透明度を大きくする（不透明にする）よう設定を行っており、ダメージ数が 100 になると透明度を 50%にするようにしている。これは完全に不透明にしてしまうとユーザーから前の景色が見えなくなってしまい運動に危険を及ぼす可能性があるためである。

このようにして Unity 上にダメージ数及びエフェクトを表示させたものを図 5.8 に示す。上の画像はダメージ数 2,下の画像はダメージ数 42 のときの様子である。上の画像に比べて下の画像は明らかに視野が赤く染まっていることがわかる。このように HoloLens2 でもユーザーの視野を半透明に赤く染めることにより、意識しなくてもダメージ数の上昇に気づくことができるようにしている。

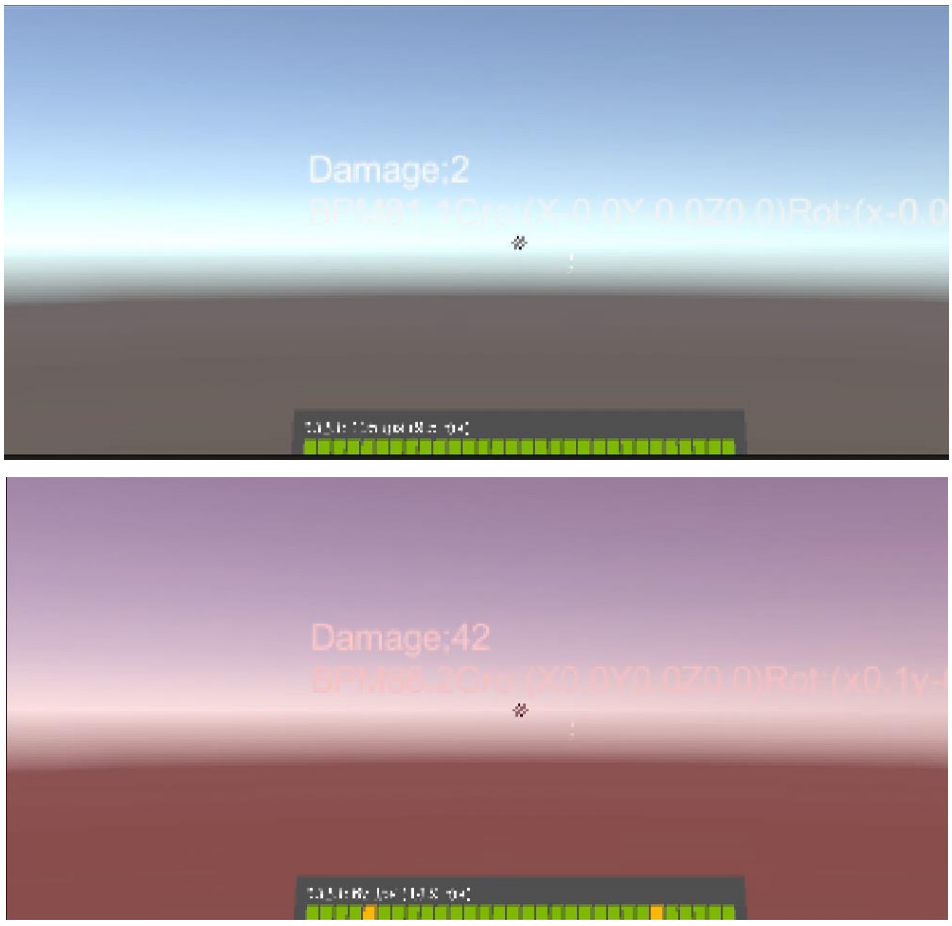


図 5.8 Unity 上にダメージ数およびエフェクトを表示させている様子
（上）ダメージ数 2　（下）ダメージ数 42

5.9 実際の使用感

本章では 5.8 までで制作したシステムを HoloLens 2、スマートフォン、スマートウォッチに実装し、実際にユーザーがどのようにシステムを使用するかを述べる。

なお、本システムでは HoloLens 2、スマートフォン、スマートウォッチがすべて同じネットワークに接続されている必要があるため、本章におけるシステムの使用では別にスマートフォンのテザリング機能を利用している。

まず初めにシステムを起動すると、図 5.9 のようにユーザーの視界にはダメージ数 0 が表示される。(図 5.9 ではスマートウォッチが正しく接続されていることを示すために心拍数等のデータを表示している)

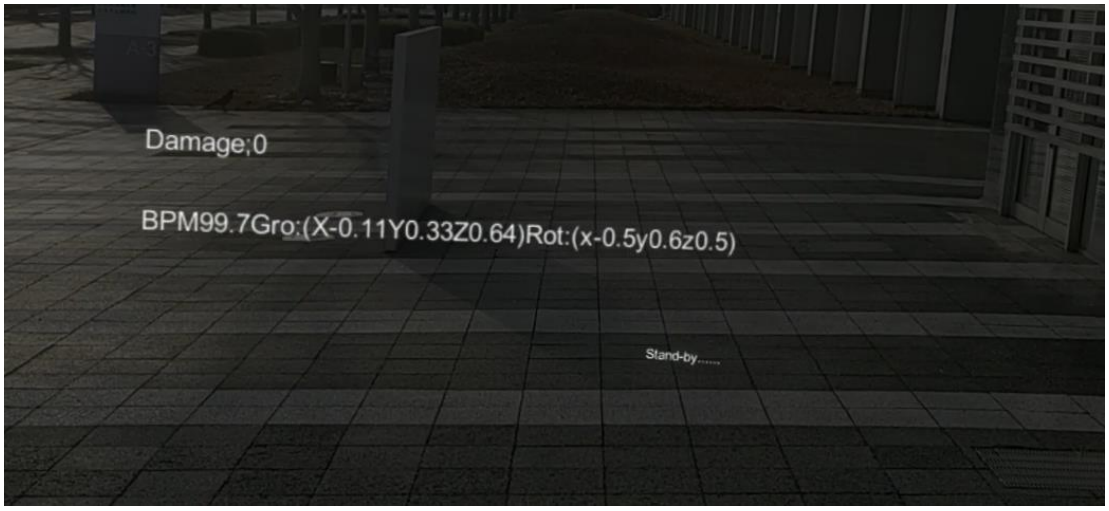


図 5.9 システム起動時のユーザーの視界

この後ユーザーが運動を開始すると、図 5.10、図 5.11 のようにダメージ数が上昇してゆき、それに伴い視界が少しずつ赤色に染まってゆく。これは熱中症のリスクが増加していることを示している。

図 5.10 運動中のユーザーの視界（ダメージ数 30 のとき）

図 5.11 運動中のユーザーの視界（ダメージ数 50 のとき）

図 5.11 からさらに運動を継続してダメージ数が 75 を超えると図 5.12 のように「そろそろ休みましょう」というメッセージが、ダメージ数が 90 を超えると図 5.13 のように「今すぐに休憩を！」という警告文が表示され、ユーザーに運動の中止を促す。

図 5.12 運動中のユーザーの視界（ダメージ数 80 のとき）

図 5.13 運動中のユーザーの視界　（ダメージ数 92 のとき）

図 5.13 の状態から休憩して心拍数が下がり、システムが「休憩に入った」と認識すると図 5.14 のようにダメージ数が下がり、エフェクトも薄くなってゆく。

図 5.14 の状態で水分補給を行ったときの様子が図 5.15 である。このように、水分補給を行うと、一気にダメージ数を 10 減らすことができる。これは休憩中だけではなく運動中に水分補給を行っても減少させることができる。

図 5.15 の直後に水を飲んだときの様子が図 5.16 である。このように、水分補給を行ってすぐ水を飲んでも時間経過によってダメージ数はわずかに減少しているがダメージ数は一気に 10 は減少しない。

ってゆく。十分に無色になったとユーザーが感じたら運動を再開する。これの繰り返

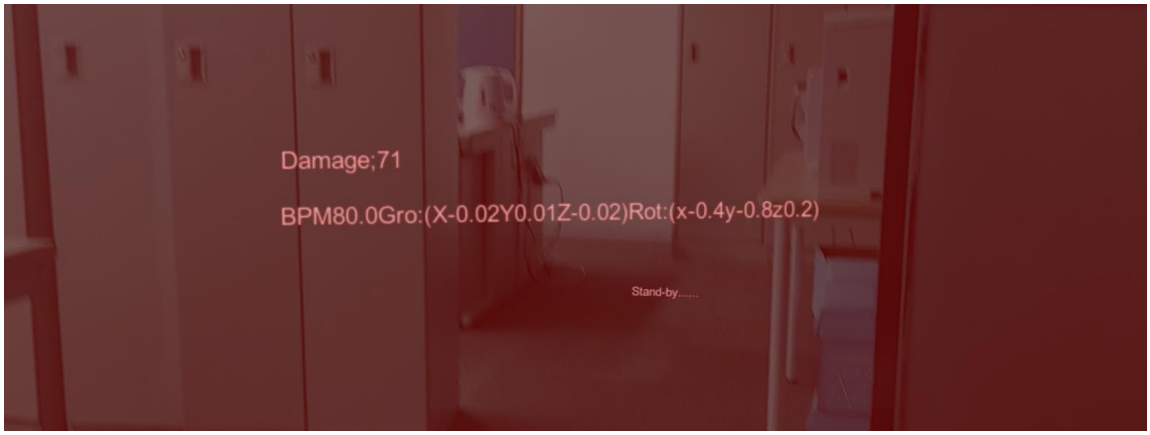


図 5.14　図 5.13 の状態のあと休憩中のユーザーの視界

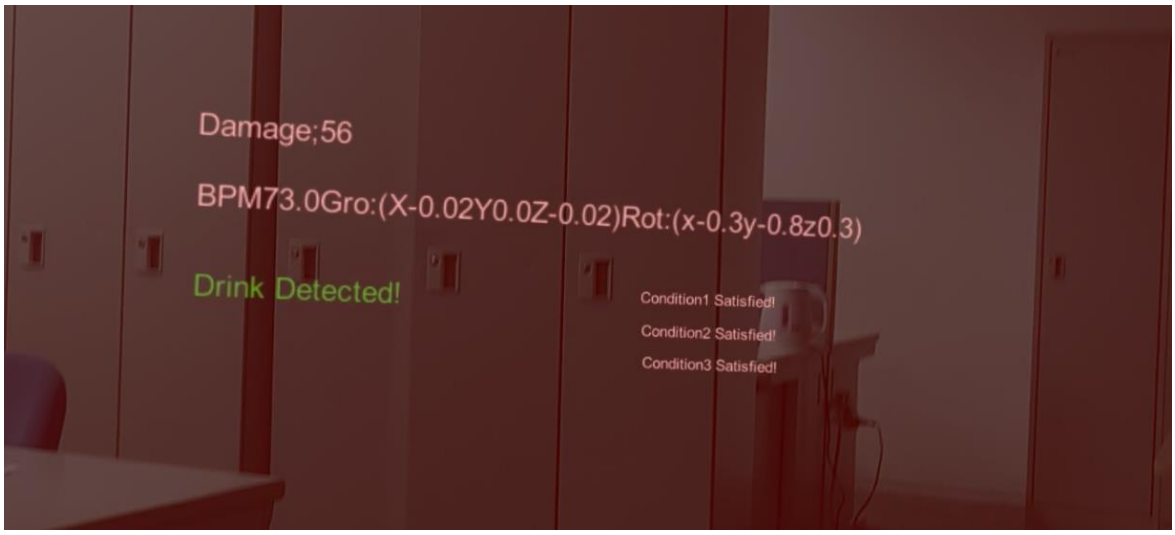


図 5.15 図 5.14 の状態からユーザーが水分補給を行った際の視界

図 5.16 図 5.15 の直後にユーザーが水分補給を行った際の視界

ダメージ数や水分補給によってダメージ数が減少すると、赤い視界が徐々に無色になしによってユーザーは安全に運動を行うことができる。

以上の手順によって、第 4 章で述べたシステムデザインを実現した。

第 6 章

まとめ

本研究ではユーザーが夏場の運動を熱中症の危険から守るべく、ユーザーが運動に集中しながら熱中症のリスクにリアルタイムに気づくことができるための手法として、AR を活用することを提案した。実装面では、スマートウォッチの心拍数による運動時間の計測、スマートウォッチのジャイロセンサーとローテーションセンサー、咽喉マイクによる嚥下音検出の 3 要素を用いた水分補給検出を行い、これらからユーザーが晒されている熱中症のリスクを示す「ダメージ数」という仮想的な値を算出した。これを基に AR を用いてエフェクトを表示させることによってシステムの実現を行った。

本システムを利用することによって、ユーザーは運動中に休憩や水分補給を適切なタイミング、回数、時間だけ行うことができるようになるため、夏場の運動を安全かつ快適に行うことが可能となる。

今後はスマートウォッチや HoloLens 2 に搭載されている様々なセンサーや機能をさらに利用し、ダメージ数の計算方法にもその日の気温や湿度、ユーザーの体調等も含める等の手法により、より安全かつ精度の高い熱中症対策システムの実現を目指す。

参考文献

[1]TANITA, 「熱中症の原因と対策」
https://www.tanita.co.jp/health/exercise/heatstroke/
（参照 2022.2.1）

[2]総務省, 「令和２年（６月から９月）の熱中症による緊急搬送状況」, [online]
https://www.fdma.go.jp/pressrelease/houdou/items/neccyuusyounennpou.pdf
（参照 2021.12.29）

[3] 環境省, 「熱中症環境保険マニュアル 2018」
https://www.wbgt.env.go.jp/heatillness_manual.php
（参照 2022.1.12）

[4] Tetsuya Yoshida, Toshimasa Takanishi, Seiichi Nakai, Akira Yorimoto, Taketoshi Morimoto, “The critical level of water deficit causing a decrease in human exercise performance: a practical field study”, European Journal of Applied Physiology 87, 529-534 (2002)

[5] 大塚製薬, 「もしも身体の水分がなくなったら」
https://www.otsuka.co.jp/nutraceutical/about/rehydration/water/dehydration-signs/
（参照 2021.12.30）

[6]Apple Store,「わたしの水 - 飲む 水 水分補給 食事管理」
https://apps.apple.com/jp/app/%E3%82%8F%E3%81%9F%E3%81%97%E3%81%AE%E6%B0%B4-%E9%A3%B2%E3%82%80-%E6%B0%B4-%E6%B0%B4%E5%88%86%E8%A3%9C%E7%B5%A6-%E9%A3%9F%E4%BA%8B%E7%AE%A1%E7%90%86/id964748094?mt=8 （参照 2021.1.27）

[7] Yanxia He, “Joining Together - An Outdoor Running Support System based on Augmented Reality” Master Thesis of Waseda IPS(2018)

[8] Zhe Li, “Food-tracker:A food recording system based on real-time food recognition and meal intake detection” Master Thesis of Waseda IPS (2018)

[9] 諸戸貴志，濱川礼，“AR による階段利用意欲向上支援システム”，情報処理学会インタラクション 2017, pp.377-382 (2017)

[10] 北園優希， 溝田直也，中島翔太, “WBGT を用いた熱中症予防アプリケーションの提案”，産業応用工学会論文誌, Vol 1, No.1 , pp.3-9 (Mar.2013)

[11] Takashi Hamatani, Akira Uchiyama, Teruo Higashino, “HeatWatch: Preventing Heatstroke Using a Smart Watch”,
2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2017, pp. 661-666

[12]大塚製薬，「スポーツ時の熱中症　対策と対処法」
https://www.otsuka.co.jp/health-and-illness/heat-disorders/casestudy/sports/
（参照 2022.1.26）

[13]「Unity-マニュアル：Unity ユーザーマニュアル 2020.3(LTS)」
https://docs.unity3d.com/Manual/index.html
（参照 2022.1.12）

[14]Microsoft, 「Mixed Reality Toolkit とは」
https://docs.microsoft.com/ja-jp/windows/mixed-reality/mrtk-unity/?view=mrtkunity-2021-05
（参照 2022.1.12）

[15]Android デベロッパー,「Android Studio の概要」
https://developer.android.com/studio/intro?hl=ja
（参照 2022.1.12）

[16]Android デベロッパー,「Sensor」
https://developer.android.com/reference/android/hardware/Sensor
（参照 2022.1.19）