筑波大学大学院博士課程

システム情報工学研究科修士論文

複数端末間連携によるウェアラブルデバイス操作手法とその応用

木下 覚

修士（工学）

（コンピュータサイエンス専攻）

指導教員　田中 二郎

2015年3月

概要

ウェアラブルデバイスには, 小型軽量故に, アプリケーションが提示できる情報量が制限されたり, 入力が困難であったりするという問題がある. 本研究は, 複数のウェアラブルデバイスを装着し, 各デバイスの画面とセンサを組み合わせる操作手法を提案する. これにより, デバイスの装着感を損なわずに個々のウェアラブルデバイスの問題点が解決することを目指す. そしてこのシステムにおける, 操作手法及びにシステム上で動作するアプリケーションを開発する事で, 本研究の有用性を示す. さらに, アプリケーションの実生活での利用を想定した利用シナリオも考案する.

目 次

図 目 次

第1章　序論

1.1　本研究の背景

我々が日々の生活で利用しているコンピューティング環境は, デスクトップ PC から始まりラップトップ, モバイルと変化してきた. そして, ウェアラブルデバイスと呼ばれる装着型のデバイスが登場しつつある. ここで, ウェアラブルデバイスの例を幾つか挙げたいと思う. まず, 頭部に装着する眼鏡型のデバイスがある. これは,The Ultimate Display[1] に端を発し, 現在においてはディスプレイのみならずカメラやタッチパネル等を備え付けたものが登場している. 次に, 手首に装着する時計型のデバイスがある. 盤面にディスプレイ及びタッチパネルを有し, 加速度センサ等複数のセンサを搭載しているものがある. 更に小型なものとして, 指に装着する指輪型のデバイスもある. ウェアラブルデバイスは, まだ一般に普及するには至っていないが, しかしモバイル端末よりも小型で軽量であるという点においても注目を集めつつあり, 今後普及する余地は十分にあると言える. 本研究では, このようなウェアラブルデバイスデバイスが普及し, 人々がそれらを複数装着して生活する現在よりも少し未来のコンピューティング環境を想定している.

1.2　ウェアラブルデバイスが抱える問題点

個々のウェアラブルデバイスの持つ問題点は大きく２つに分けて考えられる.

１つ目は, 極小画面に対する入出力問題だ. これは主に時計型のデバイスに見られる. 装着可能とするための小型軽量化によって, 搭載されるディスプレイのサイズや入力領域が限定されてしまい, 結果としてアプリケーションの提示できる情報量の減少や入力方法に制限が課せられることに繋がっている.

２つ目は, 装着部位による入出力機能の欠落だ. これは主に眼鏡型のデバイスに見られる. 眼鏡型のデバイスは, 眼前に装着する関係上, ユーザへの入力領域をデバイス自身に搭載することが困難である. これに対する解決策としては, 手元に外付けのコントローラーを用意する事や, 音声での入力が考案されている. しかしながら, 外付けのコントローラーを用いることは, ウェアラブルデバイスの装着可能なという利点を活かしきれておらず, また, 音声での入力は, 周囲の雑音に左右されることやユーザのプライバシーを損なうことから, 常に有効な入力手法であるとは言えない.

1.3　本研究の目的

本研究は, 個々のウェアラブルデバイスが持つ入出力の問題を解決するシステムの開発を目的とする. また, ウェアラブルデバイスの装着感を損なうことのないように, デバイスのみで解決するシステム構成とする. そしてこのシステムにおける, 操作手法及びにシステム上で動作するアプリケーションを開発する事で, 本研究の有用性を示す. さらに, アプリケーションの実生活での利用を想定した利用シナリオも考案する.

1.4　本研究が提案するアプローチ

我々が着目したのは個々のウェアラブルデバイスの持つ役割である. 眼鏡型のデバイスは, 広いディスプレイ領域を持つ. また, 時計型のデバイスは手首という, 人間にとって器用に動かせる部位に装着し, さらに動きを検出する為のセンサ類を持つ. 半面, 眼鏡型のウェアラブルデバイスは入力のための領域が限定され, 時計型のデバイスはディスプレイが小さく提示できる情報量が限られる. しかしながら, この２つのデバイスは同時に装着することが可能である. そこで, これらのデバイスに対して入力と出力のそれぞれの役割を分担させ, 同時に利用することで個々のウェアラブルデバイスが持つ問題を解決することができるのでは無いかと考えた.

すなわち本研究では, 眼鏡型のウェアラブルデバイス（以降 Glass）と時計型のウェアラブルデバイス（以降 Watch）の２つのウェアラブルデバイスを装着した状態での入力手法を扱う. 数あるウェアラブルデバイスの中で,Glass と Watch を選択した理由はいくつかの製品が既に発表されており (図 1.1), ウェアラブルデバイスの中では近い将来に普及する可能性が高いと考えたからだ.

そして装着した Glass と Watch を連携し, 互いのデバイスの操作面・センサ・ディスプレイを活用することで様々な入力・操作を可能にするシステムを構築していく.

図 1.1: 市場に登場しつつあるウェアラブルデバイス

1.5　本論文の構成

第１章は本章である. 第２章では, 本研究が想定するコンピューティング環境とその中での各デバイスの役割について述べる. 第３章では, 第２章で述べたコンピューティング環境における, 本研究が提案するアプローチでの入力手法のプロトタイプの概要について述べる. 第４章では, プロトタイプシステムの実装について述べる. 第５章では, 提案した入力手法を応用したアプリケーションとその利用シナリオについて述べる. 第６章では, 関連研究について述べ, ウェアラブルデバイスへの入力に関する様々なアプローチを列挙し, 本研究の位置づけを述べる. 第７章では, 本研究のまとめと今後の課題について述べる.

第2章　ウェアラブルデバイスの複数端末間連携

2.1　想定する環境と各デバイスの特徴

本研究では既に第 1 章で述べた通り, 本研究では Glass と Watch が普及し, ユーザがその 2 つを身に付けて (図 2.1) 生活するという未来環境を想定する. その中で人々は, 情報の検索やコミュニケーションといった現在モバイルデバイスで行っているタスクの一部を Glass と Watch を用いて実行する事になる. また, 各デバイスはそれぞれ次のような特徴を持つ. Glass は, ユーザの頭部に装着され, 視界を覆うように情報を提示するディスプレイを持つ. ディスプレイそのものは小型であるが, ユーザの眼前に位置するため, ユーザはあたかも大画面を見ているような感覚で情報を視認することが出来る (図 2.2). Watch は, ユーザの手首に装着され非常に自由に動かすことが可能である. また, 盤面に対してのタッチ入力や, 加速度センサによる前腕の動きの検出が可能である.

つまり,Glass は情報の出力機能に優れ,Watch は入力の機能に優れていると言える. これらの特徴を踏まえた上で, それぞれのデバイスが果たすべき役割について次節で述べる.

図 2.1: Glass と Watch を装着したユーザ

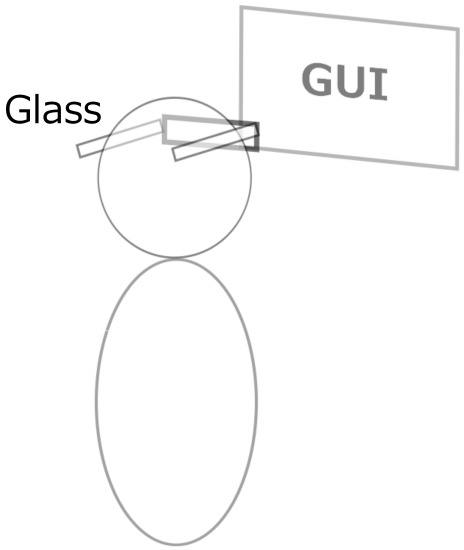

図 2.2: GUI の視認イメージ

2.2　デザインスペースと各デバイスの役割

グラフィックとポインティングデバイスを用いたGUIにおける操作では, アプリケーションが持つ機能がボタンやアイコンとなってディスプレイに表示され, ユーザがそれをポインティングデバイスで選択・実行するというのが一連の流れになる. つまり, ユーザに情報を提示する出力装置と, ユーザの命令を伝える入力装置が必要である.

ここで, 前節で述べた各デバイスの特徴を踏まえつつ役割を整理することにする. まず,Glassは広い情報提示領域を持つ. したがって, アプリケーションのGUIを表示するのに適している. しかしながら, 入力の為の手段は狭い領域へのタッチ入力や音声入力となり, 広い画面に表示されるGUIを操作するのに不向きであると考えられる. 次にWatchは, 盤面が小さく情報の提示量が限定される一方で, 手首に装着する為ある程度自由に動かすことが出来る. また, 搭載されたセンサによる動きの検出も可能である. したがって,Glassの広いディスプレイにアプリケーションのGUI表示し,Watchのセンサ等を用いてGUIの選択・実行を行うというデバイス間での連携操作が妥当であると考えられる. つまり, 本システムにおいてGlassは主に出力装置として,Watchは主に入力装置として機能することになる (図 2.3).

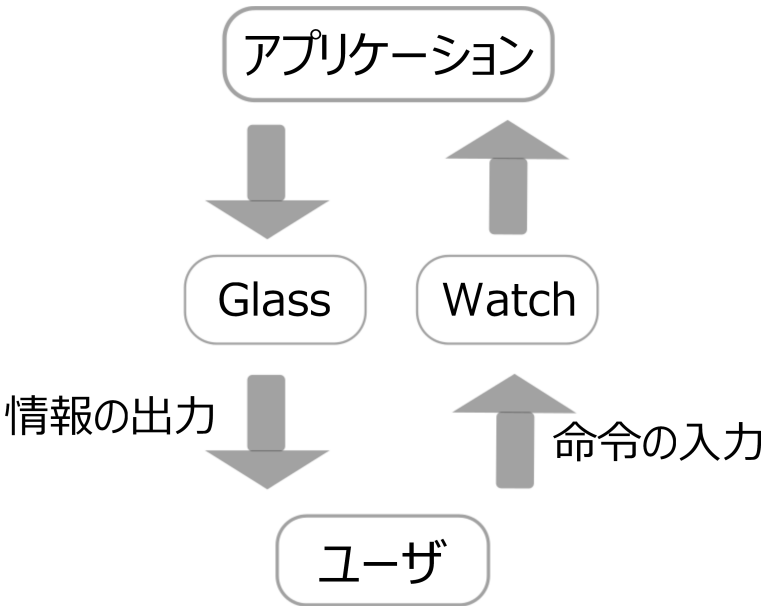

図 2.3: 入出力の流れ

2.3　インタラクション例

ここで, ２つのウェアラブルデバイスを組み合わせた, アプリケーションの起動と操作に必要となる４つのインタラクションを以下に示す.

- アプリケーションの起動
- ポインティング
- ウィジットのスクロール
- ウィジットの切り替え

2.3.1　アプリケーションの起動

アプリケーションの起動方法は２通りある. １つ目が Watch のディスプレイに表示されるアプリケーションアイコンを直接タップして起動する方法だ. これは, ユーザが直接アプリケーションを指定して起動する場合に用いる. ２つ目が Watch のディスプレイに表示される通知情報を “見る” ことによる起動だ. ウェアラブルデバイス上で動作するアプリケーションは, 時折 Watch のディスプレイに通知情報を表示する. 例えば, メールアプリケーションの新着通知や電話の着信情報がある. 見ることによる起動は,Glass のカメラを通じてディスプレイに表示された通知情報を判別することで行い, これらの通知情報に対する素早い応答を可能にする.

2.3.2　ポインティング

GUI へのポインティングは,Watch を “動かす” 事で行う (図 2.4). Glass の前に Watch を掲げ,Glass のディスプレイを通じてユーザの視界に重畳表示されるボタンやアイコンといったウィジットに対して Watch のディスプレイを重ねあわせるように手首を動かす事で各ウィジットを選択する. 選択したウィジットを実行するには, 一定時間選択状態を維持するか, もしくは Watch のディスプレイへのタッチを行うことで可能になる..

2.3.3　ウィジットのスクロール

縦長のウィジットに対してのスクロール操作は,Watch を “傾ける” 事で行う (図 2.5). Watch を装着した前腕を奥に傾け続けることででウィジットは下方向へスクロールされる. また, 手前に傾け続けることで上方向へスクロールされる. 文書ファイルや Web ページなど, 縦長の画面レイアウトは一般的である為, それらを操作するのに有効であると考えている.

図 2.4: GUI のポインティング

図 2.5: ウィジットのスクロール操作

2.3.4　ウィジットの切り替え

“傾ける” 操作は瞬間的に前腕を傾ける事でウィジットを切り替える操作へと派生する (図 2.6). これは, いくつかの画像ファイルからなるアルバムやスライド資料といった画面切り替えを多用するアプリケーションで用いる.

図 2.6: ウィジットの切り替え操作

第3章　プロトタイプシステムの概要

3.1　環境の構成

本章では, プロトタイプシステムの概要や使用した器具及びにソフトウェアについて説明する.

はじめに, 本研究が想定するコンピューティング環境ではユーザは Glass と Watch を同時に装着し生活する, したがって, プロトタイプシステムには以下の要素が必要であると考えられる.

- 眼鏡型のウェアラブルデバイスである Glass
- 時計型のウェアラブルデバイスである Watch
- デバイス間での情報の伝達手段

実際に本システムを実現する上で用いたデバイスは,Glass と Watch とスマートフォンの３つのデバイスである (図 3.1). まず,Glass には,Epson 社の Moverio BT-200 を用いた. そして,Watch には Sony 社の SmartWatch2 を用いた. さらに, スマートフォンには Sony 社の Experia Z2 を用いた. ユーザは眼前に Moverio BT-200 を装着し, 左手首に SmartWatch2 を装着する. そしてポケットもしくは鞄などでスマートフォンを携帯する. 構成にスマートフォンが含まれている理由は,SmartWatch2 がスマートフォンとの無線接続によって動作する為である. 様々なセンサ類やディスプレイを搭載しているが, 本来の想定環境ではスマートフォンの使用を想定していないためシステムでの利用は最小限に留める. したがって, ユーザがシステムを操作する際にスマートフォンを直接触ることは無い.

システムを構成する各デバイスはそれぞれ無線接続モジュールを備えるため, 連携に必要な情報はそれらの接続を介して交換することが出来る. SmartWatch2 と Experia Z2 は Bluetooth V3.0 に対応しており Serial Port Profile でペアリングしている. また Moverio BT-200 と Experia Z2 は IEEE802.11b/g/n に対応した無線接続モジュールを備えており, 無線 LAN アクセスポイントを介して無線接続される. さらに, 情報伝達手段の一部として Moverio BT-200 に搭載されているカメラと SmartWatch2 のディスプレイが使用されている.

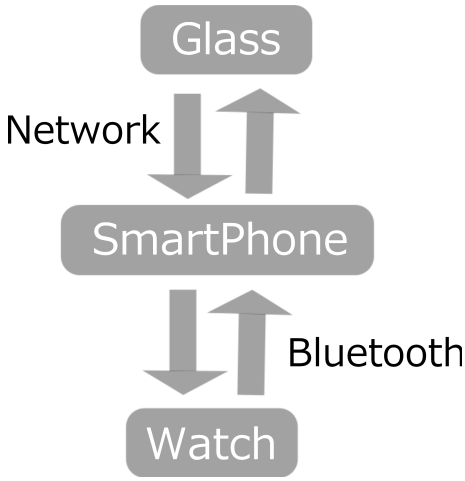

図 3.1: ハードウェアの接続イメージ

3.2 ハードウェア

システムを構成する具体的なハードウェア名に関しては, 前節で述べた. 本節では, それぞれに搭載されているセンサやディスプレイの規格 (表 3.1) とそれらがシステム中で果たす役割について述べる.

デバイスタイプ	Watch	Glass	Smart Phone
デバイス名	SmartWatch2	Moverio BT-200	Experia Z2
ディスプレイサイズ	220mm×176mm	185mm×32mm	do not use
タッチパネル	○	none	do not use
カメラ	none	○	do not use
加速度センサ	○	do not use	do not use
無線モジュール	none	IEEE802.11b/g/n	IEEE802.11b/g/n
Bluetooth	none	do not use	V3.0
外寸	185mm×170mm×32mm	42mm×41mm×9mm	147mm×73mm×8.2mm

表 3.1: ハードウェア構成

3.2.1 Glass

本システムにおいて, 眼鏡型のウェアラブルデバイスの役割を果たすのが,Epson 社の Moverio BT-200 である. Android OS が搭載されており,2015 年 1 月 13 日現在の最新バージョンは 4.0.3 である. 眼鏡のレンズにあたる部分に映像を投影する事で, ユーザに仮想的なディスプレイを視認させる. 両眼に対して映像を投影するため, 仮想的なスクリーン位置はユーザの視界の中央付近になる. 無線によるネットワーク接続が可能であり Watch との通信はおもにネットワークを経由して行う. 眼鏡の縁にカメラが搭載されており, 前方の撮影が可能である. Glass のディスプレイには, アプリケーションの GUI が投影される. GUI に対するポインティング操作は Glass のカメラで Watch を撮影することによって行う. その際,Watch 本体にはカメラ映像から検出可能な程の特徴を持たなかった為 Watch のディスプレイに特徴的な画像を表示することにしている. また, コントローラーとして外付けのタッチパッドが付属しているが, 今回想定する環境においては用いなかった.

3.2.2 Watch 及びにスマートフォン

本システムにおいて, 時計型のウェアラブルデバイスの役割を果たすのが,Sony 社の Smart-Watch2 である. ディスプレイとタッチパネル, そして 3 軸の加速度センサを搭載している. 加速度センサは, 前腕の動きを検出しスクロール操作やウィジットの切り替え操作を実現する. また, ポインティング操作に用いる為の特徴的な画像をディスプレイに表示する. ユーザは,Watch を装着した手首を Glass の前に構えてアプリケーションを操作するのが基本的な操作の為の姿勢となる. しかしながら, 動作させる為にはスマートフォンの専用アプリケーションと連動させる必要がある. その為のスマートフォンには,Sony 社の Experia Z2 を用いた. Experia Z2 は Android OS が搭載されており,2015 年 1 月 13 日現在の最新バージョンは 4.4.2 である. SmartWatch2 は Bluetooth による無線通信が可能であるが, これはスマートフォンの専用アプリケーションとの通信に使われており, その他のアプリケーションでは使用することが出来なかった. したがって, スマートフォンを経由して SmartWatch2 と Glass を無線接続することにした.

3.3 ソフトウェア

本システムを構成するソフトウェアは, 全て Android 上で動作するアプリケーションである. これは,Glass とスマートフォンに搭載されている OS に依存する. アプリケーションは Glass とスマートフォンにそれぞれ個別にインストールされる. デバイス同士が常に相互に無線接続しているため, 必要な情報をネットワークを経由したプロセス間通信によって交換している. Watch はスマートフォンにインストールされたアプリケーションと連動することで動作するため, アプリケーションのインストールは不要である.

Glass とスマートフォンにインストールされるアプリケーションには, それぞれ役割に応じた動作をする. つまり,Glass 側のアプリケーションでは GUI の描画を担当し, スマートフォン

側のアプリケーションでは Watch のセンサやディスプレイの制御を行いユーザのアプリケーション操作を可能にする. ただし, ポインティング操作に関しては Glass のカメラを使用する為 Glass 側のアプリケーションが担当する.

第4章　プロトタイプの実装

4.1　開発環境

プロトタイプシステムの開発に用いた言語, 統合開発環境, ライブラリ等を述べる. システムの開発言語は Java を用いた. また,Android アプリケーションとして実装する為に,Android SDK[1]を用いた. なお, 使用端末の関係上 Android OS のバージョンは 4.0.2 及びに 4.4.2 を想定して開発した. SmartWatch2 用のアプリケーション開発の為に Sony Add-on SDK[2]を, そして AR ライブラリとして Vuforia SDK を導入した. 統合開発環境として Eclipse 4.3[3] を使用した.

4.2　通信

プロトタイプシステムにおける,Glass と Watch の間でのプロセス間通信は TCP/IP によるソケットによって実装した. Android アプリケーションでは Java の標準ライブラリに搭載されている Socket クラスで実装可能である. その際, ネットワーク接続のパーミッションが必要な点には留意したい. また,Android3.0 以降では UI スレッドでのネットワーク通信が許可されていない為, 通信部分のプログラムは別スレッドで動かす必要がある（Watch のプログラムは UI スレッドを持たない為この限りではない）.

Glass-Watch 間でのプロセス間通信のデータフローを図 4.1 に示す. アプリケーションの通知情報は Glass から Watch へと送信され,Watch のディスプレイに表示される. アプリケーション操作の際に,Watch のセンサ値から得られる操作命令は Watch から Glass へと送信されアプリケーション操作に用いられる.

4.3　ポインティングの実装

ポインティング操作の実装では,Android アプリケーション開発環境に加えて, AR（拡張現実感）アプリケーションの為のライブラリである Vuforia SDK[4]を用いた. Watch を用いたポインティング操作の処理の流れは以下のようになる.

- Glass のカメラ画像を取得

[1] http://developer.android.com/tools/sdk/tools-notes.html
[2] http://developer.sonymobile.com/knowledge-base/sony-add-on-sdk/
[3] https://eclipse.org/
[4] https://developer.vuforia.com/

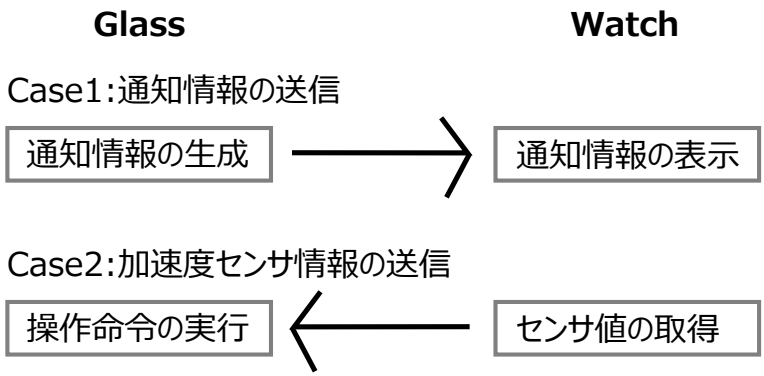

図 4.1: プロセス間通信のデータフロー

- カメラ画像中に Watch が映っているかどうかを判別
- （Watch が写っていたら）Watch の座標を取得
- Watch の座標とアプリケーション中のウィジットの座標を比較

カメラ画像に Watch が映っているかどうかの判別のために,Vuforia SDK で提供されている, 任意の画像を AR マーカー化する機能を用いた. これは, 予めマーカーとして用いたい任意の画像から特徴量を抽出しアプリケーションのデータベースに登録しておくことで, カメラ画像からマーカーを認識し, カメラからマーカーまでの 3 次元位置と傾きを取得することが出来るというものである. ただし, 認識精度を高める為にはある程度特徴量の多い画像をマーカーとして用いる必要がある. SmartWatch2 のディスプレイ部分を撮影した画像 (図 4.2) から, 特徴量の抽出を試みた結果が図 4.3 である. いくつかの特徴量を得ることが出来たが, 実際にこのマーカを用いてカメラ画像から Watch の判別を試みたがうまくいかなかった. そこで,SmartWatch2 のディスプレイに映すアプリケーションの UI を用意し (図 4.4), その UI の画像から特徴量の抽出を試みた結果 (図 4.5), うまく認識することが可能となった. また, マーカーを認識する際に取得できる, 位置情報はカメラからの相対的な 3 次元座標であるが, 今回はこの 3 次元座標をスクリーン座標への変換を行うことで 2 次元座標として利用した.

図 4.2: SW2 のディスプレイ部分

図 4.3: SW2 のディスプレイ部分の特徴量

図 4.4: アプリケーションの UI

図 4.5: UI から抽出した特徴量

“見る” という動作の検出　アプリケーションの起動トリガ等に用いる “見る”という動作は, **カメラ画像中に Watch が映っているかどうかを判別** する際に同時に識別を行っている. 今回の実装では, アプリケーション毎に Watch に表示する UI を変化させており, これによって複数のアプリケーションを判別し起動トリガとして用いた.

4.3.1 Vuforia SDK

ポインティング操作に用いる為の AR ライブラリとして,Vuforia SDK を導入した. 基本的な考え方は, 既に述べたが, ここでは実装の詳細について述べたい. Vuforia SDK によって実現したことは Watch（に表示した UI）を判別し,Glass（のカメラ）との相対位置座標を取得することである. まず,Watch に表示する UI をマーカーとしてアプリケーションに登録する必要がある. マーカーの作成は, デベロッパーサイト (`https://developer.vuforia.com/`) で画像化した UI をアップロードすることで行う. 作成されたマーカーファイルはサイト上で管理する為, 必要に応じてダウンロードしてアプリケーションに組み込むことになる. Vuforia SDK を用いた AR アプリケーションは図 4.6 の手順を実装することで実現する.

ここで,Pose Matrix とは Vuforia SDK においてマーカーを認識した際に取得出来る 3x4 の行

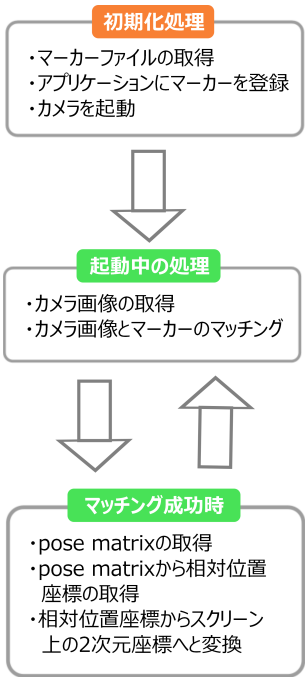

図 4.6: ポインティング操作実装の手順

列のことである. Pose Matrix の左側 3x3 にはマーカーの傾きを表す回転行列が, そして右端の 3x1 はカメラからマーカーまでの相対位置座標を意味する 3 次元ベクトルが格納されている. さて, ここで問題となるのが UI の表示されている座標はスクリーンの左上を原点とした 2 次元の**スクリーン座標系**であり,Pose Matrix から得られる座標はカメラ位置を原点とした 3 次元の**カメラ座標系**の座標である点だ. 座標系が異なる座標同士を単純に比較することは出来ない為, 座標変換を行う必要がある. なお,Vuforia SDK の Tool クラス内にカメラ座標からスクリーン座標への座標変換を行うメソッド projectPoint が実装されているためこれを用いれば容易に変換を行うことが可能である.

4.4 SmartWatch2 の制御

SmartWatch2 は, 時計型のウェアラブルデバイスであるが, デバイス上で動作するアプリケーションは全てスマートフォンと連動している. アプリケーションの GUI 画像の生成やイベントの処理はスマートフォン側が担当し,SmartWatch2 は GUI 画像の描画やタッチイベントの取得・センサ値の取得を担当している. したがって,SmartWatch2 のディスプレイやセンサ類は全てスマートフォンにインストールされたアプリケーションから制御することになる. また, スマートフォン上にインストールされる SmartWatch2 の為のアプリケーションは通常の Android アプリケーションと異なり, 専用の SDK を用いた Smart Extension としての実装が必要となる. スマートフォンにインストールされた Smart Extension はスマートコネクトというアプリケー

ションを介して SmartWatch2 と連動することになる (図 4.7).

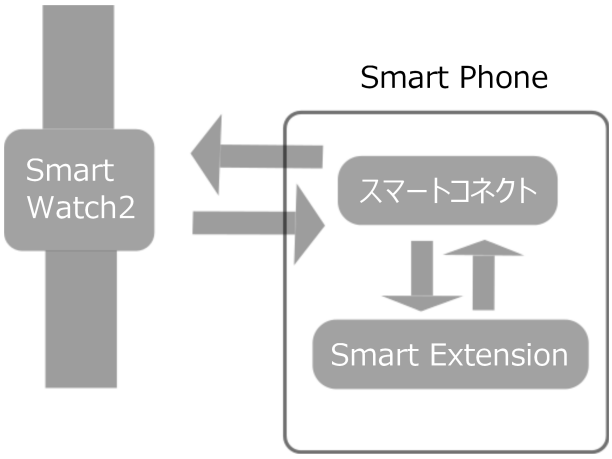

図 4.7: SmartWatch2 とスマートフォンの連動イメージ

Smart Extension による SmartWatch2 の制御の流れは, 以下のようになる.

- Smart Extension の起動
- UI の生成と画像化
- UI 画像を送信し Watch ディスプレイに描画
- Watch ディスプレイのタッチ入力及びに, センサ情報の更新待ち
- ディスプレイのタッチイベント, センサ値の送信

Smart Extension の実体は Android にインストールされたバックグラウンドで動作するサービスである. インストールされた Smart Extension はスマートコネクトに登録され, アプリケーションのアイコンが Watch のディスプレイに表示される. Smart Extension はアプリケーションアイコンのタップによる起動かインストール時に起動して常駐するという２通りの動作方法がある. Glass のアプリケーションから通知情報を受け取って表示するという動作を実現する為に今回は常駐タイプの実装を採用した. UI の生成にはレイアウトファイルを読み込んで生成する, プログラム中でレイアウトを定義して生成する,Canvas に図形やテキストを描画するといったやり方が考えられる. ただし,SmartWatch2 のディスプレイサイズが小さい為,Android のレイアウトをそのまま適用することは困難である. また, 生成した UI は Watch に送信する為に Bitmap に変換する必要がある. ディスプレイのタッチイベントはタッチ入力が行われた際に, イベントの種類, TOUCH DOWN もしくは TOUCH UP の情報と指の触れた座標が取得できる. マルチタッチでの入力には対応していない点とディスプレイ上で指を動かした際にはタッチイベントが取得できない点には注意が必要である（フリックの取得はタッチイベントとは別のイベントで取得可能であるが本システムの実装では用いなかった為説明は省略す

る）. SmartWatch2 では, 加速度センサと照度センサの２つのセンサが搭載されており, アプリケーションの中で利用可能である. センサ値は精度が選択できる (Low or Middle or High). 実装で用いた加速度センサについては次節で説明する.

4.5　SmartWatch2 の加速度センサ

スクロール操作やウィジットの切り替え操作に用いる, 前腕を傾ける動作は Watch の加速度センサを用いた. Watch の加速度センサは 3 軸であり (図 4.8), プロトタイプシステムで用いる各操作に対応する前腕の傾きの検出の為に Y 軸方向の値を用いた.

図 4.8: Watch の加速度センサの軸方向

SmartWatch2 の制御については既に述べた通り,Smart Extension として実装する必要がある. Sony Add-on SDK がその実装の為に導入したライブラリである. SDK によって提供される実装の為のフレームワークにはセンサ類の値が更新された時に呼ばれるコールバックメソッドが用意されており, そのメソッドをオーバーライドする事でセンサの値を用いた様々な実装が可能になる. 今回は, このメソッドをオーバーライドすることで加速度センサの値を用いて前腕の動きを検出した.

プロトタイプシステムで採用したインタラクションのうち, スクロール操作とウィジットの切り替え操作の２つは前腕の動きを用いる. スクロール操作は, 前腕を自然な位置よりも傾け続けた状態である. その際のユーザの前腕の動きは以下のようになる.

- 前腕を傾ける
- 傾けた状態を維持する
- ウィジットがスクロールされる
- 傾けた前腕を元の位置に戻す

つまり, 自然な位置にある時の加速度センサの Y 軸の値と比較して一定の値以上異なる状態を一定時間維持した場合が, スクロール操作であるといえる. また, ウィジットの切り替え操作は前腕を素早く動かす動きである. 切り替え操作の際の前腕の動きは以下のようになる.

- 前腕を傾ける
- すぐに傾けた前腕を元の位置に戻す
- ウィジットが切り替わる

これは, 自然な位置にある時の加速度センサの Y 軸の値と比較して一定の値以上異なる状態から, 一定時間以内に復帰した場合が該当する. よって, スクロール操作と切り替わる操作は, 傾けた状態が維持される時間によって区別可能である. スクロール操作が成立する条件は傾き始めた時刻を startTime 現在時刻を currentTime とするとしきい値 th を用いて以下の式で表される.

$$currentTime - startTIme > th \tag{4.1}$$

4.6 アプリケーションのウィジット

Android アプリケーションで利用可能なウィジットはタッチ操作を基本としている為, 全てを本システムでそのまま用いることは出来なかった. したがって, アプリケーションで利用する為のウィジットとして, ボタンとスクロールウィジット切り替えのウィジットを実装した.

4.6.1 ボタンウィジット

ボタンは, テキストを表示し選択された際に特定の操作を実行する四角い形状のウィジットである. ポインティング操作によって選択することが出来る. 通常半透明であるが, 選択された際には不透明に変化することで選択された事を視覚的にフィードバックする.

4.6.2　スクロールウィジット

テキストや画像を格納する事が出来る縦長のウィジットである. ウィジットの横の長さは実装時に指定し, テキストは右端で折り返される. また, 画像は横の長さに沿って拡大縮小される. スクロール操作によって上下方向にスクロール出来る.

4.6.3　切り替えウィジット

複数の画像を格納することが出来るウィジットである. 複数枚の画像を格納した際に, 実際に画面上に大きく表示されるのは１枚のみとなる. しかし, 切り替え操作を行うことでウィジットに表示する画像を切り替えることが出来る.

第5章　アプリケーションと利用シナリオ

本研究では提案した入力手法, 及びにそれらを実装したプロトタイプシステムに加えて, 応用として本システム上で動作するアプリケーションの実装を行った. 本章では実装したアプリケーションとその利用シナリオについて述べる.

5.1　メール閲覧アプリケーション

本システム上で動作するアプリケーションとして, メールの閲覧アプリケーションを実装した. 本アプリケーションでは, メールの受信を通知し受信したメールの本文及びに添付された画像ファイルの確認が出来る.

5.2　利用シナリオ

本アプリケーションの利用シナリオを述べる. 今回は, 大学生が日常生活の一場面で利用するというケースをシナリオとした.

5.2.1　通学電車にて

A さんは,T 大学に通う大学生である. T 大学は,A さんの自宅から電車で 1 時間の所にあり,A さんは平日は毎日大学に通っている. A さんは, 自宅ではパソコンとモバイル端末を使い分けており, 主な用途は課題のレポートの作成と研究室でのメールのやり取りである. そんな A さんだが, 通学の為の移動時は Glass と Watch を用いている. 身に付ける事が出来る為, 携帯しやすく使いたいときにすぐ起動出来るからだ.

ある日, 電車のつり革に掴まりながら窓の外を眺めていると Watch のバイブレーションが振動していることに気がついた. 盤面に目をやるとそこには, メールの受信を知らせる通知が表示されていた. 気になった A さんは, そのままメールを閲覧することにした. Glass のディスプレイにアプリケーションが映し出されメールが開封された. メールには, 研究室の同期から国際会議の会場についたことを知らせる旨が書かれており, 同時に数枚の写真が添付されていた. アプリケーションを操作して写真を見ると, どうやら会場となっているらしい格式高そうな会議場と A さんが尊敬する教授が写った写真が含まれていた. 改めて, メールを読み返すとその教授が招待講演を行うことと, その講演がインターネット経由で配信されるらしいということ

が書いてあることに気がついた. A さんは貴重な情報をもたらしてくれた同期に感謝しつつ, 今日は研究室に着いたらまずはこの配信を見ることから始めようと心に決めたのであった.

5.3 アプリケーションの操作フロー

アプリケーションの起動からの一連の操作の流れを操作フローとして示した (図 5.1). アプリケーションはバックグラウンドでメールの受信を待ち受け, 受信があった時にユーザに通知を行う. メールを受信したという通知情報が Watch のディスプレイに表示され, その通知情報をユーザが見ることでアプリケーションがフォアグラウンドに切り替わる. フォアグラウンドとなったアプリケーションでは受信したメールの本文と画像ファイルは Glass のディスプレイにそれぞれ表示される. メール本文は縦長のウィジットに格納されており, 全文を表示しきれない場合でもスクロール操作によって閲覧出来る. また, 複数の画像ファイルが添付されていた場合はウィジットの切り替え操作によって表示する画像ファイルを適宜切り替える事が可能である. さらに, 外出時の利用を想定し, 全ての操作は片手での操作に対応している.

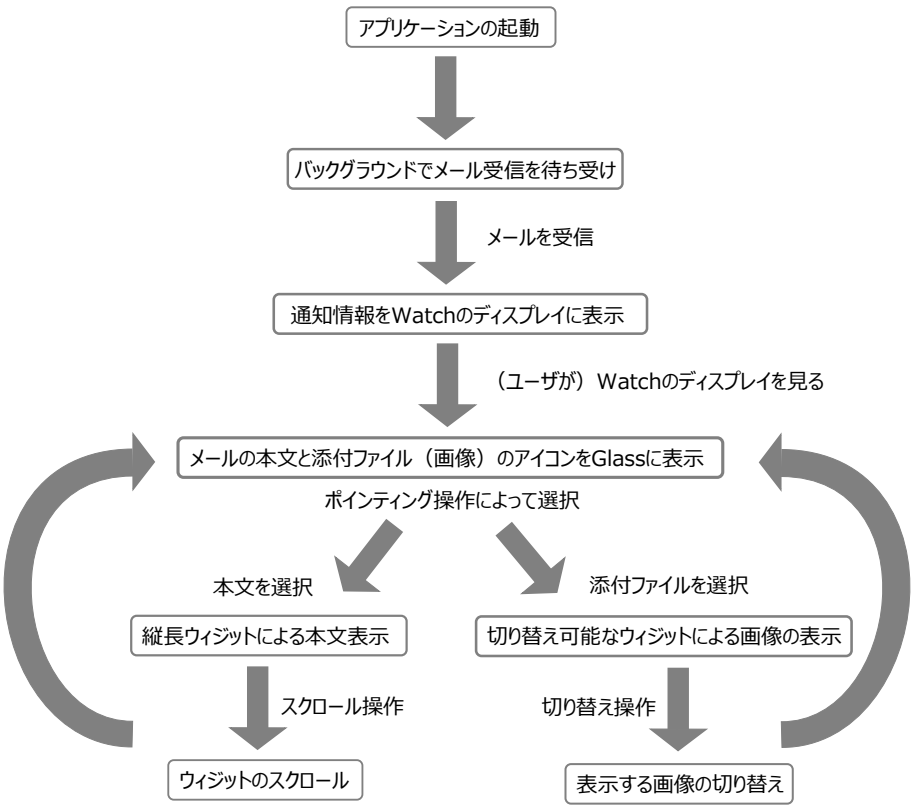

図 5.1: アプリケーションの操作フロー

5.4 アプリケーションの実装

アプリケーションの実装に際して別途用意した環境を述べる. メールアカウントには gmail[1] を用いた. また, アカウント認証及びに, メールの受信の為には java mail-android [2] を用いた.

[1] https://www.gmail.com/
[2] https://code.google.com/p/javamail-android/

第6章　関連研究

6.1　ウェアラブルデバイスとその操作手法の研究

ウェアラブルデバイスとその操作手法の研究は多数存在する. ここでは, デバイスの形状毎に分類し列挙していく. これらのウェアラブルデバイスの操作手法のアプローチは, ひとつのウェアラブルデバイスを拡張し, 様々な操作を可能にしてきたと言える. しかしながら, ひとつのデバイスを拡張し続ける事は, デバイスの大型化に繋がる恐れがある. 我々は装着感の観点からウェアラブルデバイスの大型化は好ましくないと考える. 本研究では, 複数のウェアラブルデバイスを連携することで個々のデバイスを拡張すること無く, 様々な入力を可能にしている.

6.1.1　時計型デバイス

GestureWrist and GesturePad[2] では, 手首に装着し, 加速度センサとピエゾアクチュエータを用いてハンドジェスチャや前腕の動きを認識することが出来る. Abracadabra[3] は, 指に装着した磁力を手首に取り付けたデバイスで認識することで, 画面へのタッチ入力以外を可能にする. TouchSense[4] は, 指先にセンサを取り付け時計型ウェアラブルデバイスのタッチインタラクションを拡張している. また Skin Buttons[5] は, レーザープロジェクタを搭載しデバイスの周辺にウィジットを投影することで, 盤面以外でのタッチインタラクションを実現した. その他, 時計型デバイスへの操作手法として, ベゼルを用いたもの [6][7] やバンド [8] を用いたものなどが提案されてきた.

6.1.2　眼鏡型デバイス

眼鏡型のウェアラブルデバイスに対する入力としては, モバイル端末をコントローラーとして用いる研究 [9] やカメラ画像を用いたハンドジェスチャによる入力 [10] がある. Google Glass[11] では, 音声による入力の他に眼鏡の縁にタッチパネルを搭載しシンプルなタッチ入力が可能である. また,Googla Glass に内蔵された IR センサを用いてまばたきを検出するという試みもなされている [12]. Mime[13] は,RGB カメラと３つのフォトダイオードを用い３ D ハンドジェスチャを識別する. Marcos らは, 顔をなぞって入力を行う Hand-To-Face インタラクションを提案した [14].

6.1.3　指輪型デバイス

Nenya[15] は指輪型のウェアラブルデバイスである. 指輪に埋め込まれた磁力を, 手首に装着した磁気センサによって計測することで, ジョグダイヤル方式の入力を行う. また,iRing[16] はフォトリフレクタセンサを用いて指の動きを認識し, プッシュとストロークによる入力を行う. EyeRing[17] は小型カメラを搭載し, ビジョンベースでの操作を行う. LightRing[18] では, フォトリフレクタセンサとジャイロセンサを組み合わさせることで平面上での指の動きをトラッキングすることが出来る.

6.1.4　その他の形状のデバイス

Tamaki らは, 耳に掛ける形状のウェアラブルデバイス [19] を開発した. レーザープロジェクタによる視覚的フィードバックとカメラによるハンドジェスチャの認識を行う. ShoeSense[20] は, 靴に載せたデプスカメラによるジェスチャインタラクションの可能性を示した. ShoeSoleSense[21] は靴底に敷き詰めたセンサによって足底の接地の仕方による様々な入力を実現した. OmniTouch[22] は肩に載せたピコプロジェクタとデプスカメラを用いて手のひらに仮想的なタッチディスプレイを作り出す.

6.2　デバイス間連携の研究

デバイス間連携の研究はいくつか行われている. ここでは連携の目的に合わせて列挙していく. 本研究では, ウェアラブルデバイスのみを用いたデバイス間連携により様々な入力を実現する. これは, 他の研究とは異なりウェアラブルデバイスのみで構成されるコンピューティング環境において有効な操作手法であると考えられる.

6.2.1　複数デバイス間でのデータ共有

Pick and Drop[23] では複数のコンピュータ間での情報の移動をペンと画面のタッチインタラクションによって実現している. 近接する 2 つのタブレットで同時にバンプジェスチャを行うことで, タブレット間で画像を共有する研究 [24] もある. 同期的なジェスチャによるデバイス間連携としては他にも, タンジブルインタフェースによるもの [25] がある. アプリケーション上で, 近くにあるコンピュータとの空間的な位置関係を図示することで, データ共有を容易にしようという研究 [26] もある. Gradual engagement[27] は, デバイス間の関係と共有の為のインタラクションを示す事で,“どのデバイスとどうやって情報を交換するのか分からない” という状況の解決を試みた.

6.2.2　インタラクションの拡張

Touch Projector[28] は通常近づいて触る事が困難な大画面に対するタッチインタラクションをユーザが手に持ったモバイルデバイスのビデオ映像を通して行う事が出来る. また, モバイルデバイスと大画面とのデータ交換を行うことも出来る. Facet[29] は複数の時計型ウェアラブルデバイスの盤面を組み合わせることで, 複雑なマルチタッチによる入力を可能にした. Duet[30] は Watch とスマートフォンを連携し, 個々のデバイスの操作面やセンサを用いてタッチパネルだけでは実現出来ないタッチインタラクションを実現している.

第7章　まとめと今後の課題

本研究では, 個々のウェアラブルデバイスが持つ入出力の問題点に対して, 装着した複数のウェアラブルデバイス毎に役割を分担させつつ同時に利用するというデバイス間連携のアプローチを取ることで解決を図った. また, 考案したアプローチのプロトタイプシステムを構築し, システム上でのインタラクションの提案及びに応用例としてメールの閲覧アプリケーションを実装した. さらに, 実生活での利用を想定した利用シナリオを考案した.

今後の課題としては, 更なる活用先を想定したインタラクションの追加が挙げられる. これは, 実際のアプリケーションでは文字入力やジェスチャ・マルチタッチといった複雑な操作を行う場面も多く見られる為, それらに対応した入力を用意する必要がある. また, 様々な用途に対応するためにメールの閲覧アプリケーションの他に, 新たなアプリケーションを設計し実装する事も重要である. 更に, 提案したインタラクションが有効であるかどうかを定性的・定量的に測定する必要もある.

謝辞

本論文を執筆するにあたって，指導教員である田中二郎先生をはじめ，三末和男先生，高橋伸先生およびに志築文太郎先生にはゼミや面談を通して，丁寧なご指導と貴重なご意見を頂きました．心より感謝申し上げます．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，研究生活全体にわたって数多くのご指摘やご意見を頂きました．厚く御礼申し上げます．特に，上級生の方々には議論の機会を快く設けていただき，その中で数々の発見と研究テーマの深まりがあったことについて感謝の念を禁じえません．大変ありがとうございました．

参考文献

[1] Ivan E. Sutherland. The ultimate display. In *Proceedings of the IFIP Congress*, pp. 506–508, 1965.

[2] Jun Rekimoto. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. In *ISWC 2001*, pp. 21–27, 2001.

[3] Chris Harrison and Scott E. Hudson. Abracadabra: Wireless, high-precision, and unpowered finger input for very small mobile devices. *UIST '09*, pp. 121–124, 2009.

[4] Da-Yuan Huang, Ming-Chang Tsai, Ying-Chao Tung, Min-Lun Tsai, Yen-Ting Yeh, Liwei Chan, Yi-Ping Hung, and Mike Y. Chen. Touchsense: Expanding touchscreen input vocabulary using different areas of users' finger pads. *CHI '14*, pp. 189–192, 2014.

[5] Gierad Laput, Robert Xiao, Xiang 'Anthony' Chen, Scott E. Hudson, and Chris Harrison. Skin buttons: Cheap, small, low-powered and clickable fixed-icon laser projectors. *UIST '14*, pp. 389–394, 2014.

[6] Daniel Ashbrook, Kent Lyons, and Thad Starner. An investigation into round touchscreen wristwatch interaction. *MobileHCI '08*, pp. 311–314, 2008.

[7] Ian Oakley and Doyoung Lee. Interaction on the edge: Offset sensing for small devices. *CHI '14*, pp. 169–178, 2014.

[8] Simon T. Perrault, Eric Lecolinet, James Eagan, and Yves Guiard. Watchit: Simple gestures and eyes-free interaction for wristwatches and bracelets. *CHI '13*, pp. 1451–1460, 2013.

[9] Youngjin Hong, Sanggoog Lee, Yongbeom Lee, and Sangryong Kim. Mobile pointing and input system using active marker. *ISMAR '06*, pp. 237–238, 2006.

[10] 加茂浩之, 田中二郎. Air surface : 拡張現実上の平面を用いたインタフェース. 全国大会講演論文集, pp. 299–301, 2011.

[11] Google. Google glass. `http://www.google.com/glass/start/`.

[12] Shoya Ishimaru, Kai Kunze, Koichi Kise, Jens Weppner, Andreas Dengel, Paul Lukowicz, and Andreas Bulling. In the blink of an eye: Combining head motion and eye blink frequency for activity recognition with google glass. *AH '14*, pp. 15:1–15:4, 2014.

[13] Andrea Colaço, Ahmed Kirmani, Hye Soo Yang, Nan-Wei Gong, Chris Schmandt, and Vivek K. Goyal. Mime: Compact, low power 3d gesture sensing for interaction with head mounted displays. *UIST '13*, pp. 227–236, 2013.

[14] Marcos Serrano, Barrett M. Ens, and Pourang P. Irani. Exploring the use of hand-to-face input for interacting with head-worn displays. *CHI '14*, pp. 3181–3190, 2014.

[15] Daniel Ashbrook, Patrick Baudisch, and Sean White. Nenya: Subtle and eyes-free mobile input with a magnetically-tracked finger ring. *CHI '11*, pp. 2043–2046, 2011.

[16] Masa Ogata, Yuta Sugiura, Hirotaka Osawa, and Michita Imai. iring: Intelligent ring using infrared reflection. *UIST '12*, pp. 131–136, 2012.

[17] Suranga Nanayakkara, Roy Shilkrot, Kian Peen Yeo, and Pattie Maes. Eyering: A finger-worn input device for seamless interactions with our surroundings. *AH '13*, pp. 13–20, 2013.

[18] Wolf Kienzle and Ken Hinckley. Lightring: Always-available 2d input on any surface. *UIST '14*, pp. 157–160, 2014.

[19] Emi Tamaki, Takashi Miyaki, and Jun Rekimoto. Brainy hand: An ear-worn hand gesture interaction device. *CHI EA '09*, pp. 4255–4260, 2009.

[20] Gilles Bailly, Jörg Müller, Michael Rohs, Daniel Wigdor, and Sven Kratz. Shoesense: A new perspective on gestural interaction and wearable applications. *CHI '12*, pp. 1239–1248, 2012.

[21] Denys J. C. Matthies, Franz Müller, Christoph Anthes, and Dieter Kranzlmüller. Shoesolesense: Proof of concept for a wearable foot interface for virtual and real environments. *VRST '13*, pp. 93–96, 2013.

[22] Chris Harrison, Hrvoje Benko, and Andrew D. Wilson. Omnitouch: Wearable multitouch interaction everywhere. *UIST '11*, pp. 441–450, 2011.

[23] Jun Rekimoto. Pick-and-drop: A direct manipulation technique for multiple computer environments. *UIST '97*, pp. 31–39, 1997.

[24] Ken Hinckley. Synchronous gestures for multiple persons and computers. *UIST '03*, pp. 149–158, 2003.

[25] David Merrill, Jeevan Kalanithi, and Pattie Maes. Siftables: Towards sensor network user interfaces. *TEI '07*, pp. 75–78, 2007.

[26] Gerd Kortuem, Christian Kray, and Hans Gellersen. Sensing and visualizing spatial relations of mobile devices. *UIST '05*, pp. 93–102, 2005.

[27] Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. Gradual engagement: Facilitating information exchange between digital devices as a function of proximity. *ITS '12*, pp. 31–40, 2012.

[28] Sebastian Boring, Dominikus Baur, Andreas Butz, Sean Gustafson, and Patrick Baudisch. Touch projector: Mobile interaction through video. *CHI '10*, pp. 2287–2296, 2010.

[29] Kent Lyons, David Nguyen, Daniel Ashbrook, and Sean White. Facet: A multi-segment wrist worn system. *UIST '12*, pp. 123–130, 2012.

[30] Xiang 'Anthony' Chen, Tovi Grossman, Daniel J. Wigdor, and George Fitzmaurice. Duet: Exploring joint interactions on a smart phone and a smart watch. *CHI '14*, pp. 159–168, 2014.

付録

想定する未来環境

本研究で想定する未来環境では, 我々はウェアラブルデバイスを複数装着し生活する. そのような環境では, 外出時等ウェアラブルデバイスのみを用いて, メールや地図といったアプリケーションを操作する事が考えられる. つまり, ウェアラブルデバイスのみで構成された操作手法を模索する必要がある. しかしながら, そのような環境で我々が利用するコンピューティング環境はウェアラブルデバイスだけでは無い. これは, ラップトップやモバイルデバイスが普及した時と同様で, 新たなコンピューティング環境はそれ以前のコンピューティング環境を完全に上書きするものではなく, あくまでも一部を肩代わりしたり, 新しい事を可能にしたりするものであるからだ.

また, コンピューティング環境の変化に伴い, デバイスの操作手法も変化してきた. デスクトップ PC では, マウスとキーボードを用いての操作が一般的であった. ラップトップにおいては, 新たなポインティングデバイスとしてタッチパッドを用いるようになった. そして, スマートフォンやタブレット端末といったモバイルデバイスにおいては, 直接ディスプレイを触って操作するようになった. これらは, デバイスに合わせた変化であり, また平面に対する入力と言える. しかしながら, 身につけられる程小型・軽量なウェアラブルデバイスにおいては, 操作面が限定される為, 従来のような平面に対する入力は困難であるという問題がある. その問題を解決するアプローチの一つに音声入力が挙げられる. 音声入力は人が発する音声をマイクを使って認識し, デバイスを操作するというものである. ただし, 音声によるデバイス操作は, こと公共の場所においては周囲への配慮や羞恥心から使用が阻害される恐れがある. したがって, 従来のウェアラブルデバイスに対するアプローチとは異なった入力手法を模索していく必要があると考えられる.