卒業論文 2004 年度 (平成 16 年度)

ホームコンピューティングにおける手指を用いた入力インタフェースの研究

指導教員

慶應義塾大学環境情報学部

徳田　英幸

村井　純

楠本　博之

中村　修

南　政樹

慶應義塾大学 総合政策学部

山崎俊作

卒業論文要旨 2004 年度 (平成 16 年度)

ホームコンピューティングにおける手指を用いた 入力インタフェースの研究

本研究では，ホームコンピューティングにおいて，利用者が様々なコンピュータを容易に操作するための入力インタフェースを実現する．これを実現するため，本論文では手指を用いた身振りインタフェースを提案し，身体を利用した統一的な操作系に基づく入力インタフェースを開発した．また，統一的な操作系に基づく入力インタフェースに対応したアプリケーションを，容易に構築するためのアプリケーションフレームワーク SUI を開発した．

ホームコンピューティングでは，コンピュータは従来のパーソナルコンピュータの形状ではなく，情報家電や家具，壁といったものに埋め込まれるのに適した形状で利用者の身の回りに遍在する．これらの様々な形状のコンピュータは連携しながら動作するため，高性能化，多機能化がすすむことが想定される．一方で，テーブルトップでの利用が主であったコンピュータを実生活に持ち込んだことにより，GUI や従来のリモコンがホームコンピューティングにおける入力インタフェースとして一般的である．しかし，GUI やリモコンは利用者の置かれている状況などの物理的要因がほとんど考慮されていないため，コンピュータの知識が乏しい利用者が想定されるホームコンピューティングでの入力インタフェースとして適切ではない．また，様々なコンピュータに対して一貫した操作系を持たず，複数のコンピュータへの協調動作の指示についても考慮されていない．

本研究では操作方法として手指によるジェスチャを用いることで, 操作対象への直接操作を実現する．また，操作方法に用いるジェスチャをさす，うつ，つまむ，はなす，まわすの 5 つに限定することで，利用者が習得しやすく，統一的な操作系にもとづいた操作インタフェースを実現する．また，本研究で提案する SUI フレームワークを用いることで，ホームコンピューティングのアプリケーションを容易に統一的な操作体系に対応させることができる．

本論文では，まず本研究が想定するコンピューティング環境であるホームコンピューティングについて解説し，従来の入力インタフェースを整理する．ついで，身振りによる入力インタフェースの関連研究をとりあげる．また，関連研究の問題点を指摘し，これを解決するための本研究のアプローチを述べる．ついで，SUI フレームワークを提案し，その設計と実装を行う．最後に定量的評価をもとに本研究の有用性の証明と今後の改良の一助を得る．

慶應義塾大学 総合政策学部

山崎 俊作

目次

図目次

表目次

第 1 章

序論

1.1 本研究の背景

近年，短距離無線通信技術の普及，プロセッサの計算処理能力の向上，コンピュータの小型化などによりテーブルトップでの利用が主であったコンピュータが，モバイル機器や情報家電機器といった形で実生活全般にまで利用されるようになった．これらのコンピュータが持つネットワーク接続機能を利用し，利用者がコンピュータの存在を意識することなくコンピュータを利用可能にするユビキタスコンピューティング [17] の研究開発が盛んである．ユビキタスコンピューティング環境では様々なコンピュータが連携しながら動作するため，高性能化，多機能化がすすむ反面，従来型のリモートコントローラ（以下，リモコン）やスイッチでは対応できないほど操作が複雑になることが想定される．このため，RFID や小型無線センサネットワーク，画像解析技術などのセンシング技術を応用して，利用者の行動や利用者の身の回りの事象を検知し，利用者の置かれた状況にあわせて自律的に動作するコンテクストアウェアアプリケーションが注目されている．

コンテクストアウェアアプリケーションでは環境に偏在するセンサ群から取得されたデータ群を解析し，関連性，規則性を見いだすことで利用者の行動を把握する必要がある．しかし，利用者の行動を完全に把握することは極めて困難であるため，利用者が能動的にコンピュータを操作しなければならない場合がある．例えば，現在，もっとも普及している自律的に動作する機械に自動ドアがある．自動ドアで利用者の検知に用いられるセンサは赤外線センサが一般的であり，検知精度は高いが失敗することもある．センサが利用者の検知に失敗した場合，利用者は一度後退してからドアの前に立ちなおしたり，センサに向けて手をかざしたり自動ドアを動作させるために能動的な行動をとる光景がよく見られる．コンテクストアウェアアプリケーションでは，このような単純な行動だけでなく，利用者 A が t 時間就寝しているというような複雑な状況も把握しなければならない．しかし，利用者の行動や利用者の置かれた状況すべてを過不足なく完全に把握することはきわめて困難であるといえる．したがって，コンテクストアウェアアプリケーションでは対応しきれない場面において，利用者がコンピュータを操作するための入力インタフェースが重要になってくる．

一方で，テーブルトップでの利用が主であったコンピュータを実生活に持ち込んだことにより，従来の WIMP（Window・Icon・Menu・Pointer）を用いた GUI が，依然としてモバイル機器や情報家電への入力インタフェースとして一般的である．WIMP による GUI は利用者の置かれている状況などの物理的要因がほとんど考慮されていないため，コンピュータの知識が乏しい利用者が想定される実生活でのコンピュータへの入力インタフェースとして適切ではない．このため，物理的あるいは身体的な要素を生かしたインタフェースの実現が望まれている．このような分野は実世界指向インタフェースや Augmented Reality [1]，Tangible User Interface [7] などと呼ばれ，研究開発が行われている．しかし，様々なコンピュータに対して一貫した

操作系を持たず，また，複数のコンピュータへの協調動作の指示について考えられていない．

1.2 本研究の目的

本研究の目的はホームコンピューティングにおいて，利用者の身体を利用した統一的な操作系に基づく入力インタフェースを実現することである．ホームコンピューティングにおける操作可能なコンピュータ（以下，操作対象）は多機能かつ多種に及ぶため，利用者がそれらすべての操作方法を習得することは困難である．しかし，入力インタフェースを一般化，単純化することは可能なため，利用者は統一的な操作系による入力インタフェースによって，特別な訓練なしに操作対象を操作できる．

これに加え，利用者の身体を利用することで，WIMP による GUI やリモコンよりユーザが直接操作対象を制御している感覚（以下，直接感）の高い入力インタフェースを目指す．WIMP による GUI やリモコンでは操作対象を操作するために，「ボタン A（主語）」を「3 回押す（述語）」といったシンタックスを意識しなければならなかった．利用者は自らの身体を利用することで，シンタックスを意識せずに操作対象を操作できる．

また，本研究では，操作対象によって異なる機能の多様性を隠蔽し，統一的な操作系による操作を実現する SUI Framework を提案する．SUI Framework では，操作対象の機能，および入力インタフェースのコマンドを抽象化し，操作対象と入力インタフェース間の通信機能を提供する．これにより，ホームコンピューティングの開発者は容易に統一的な操作系に対応した入力インタフェース，アプリケーションを作成できる．

1.3 本論文の構成

本論文は本章を含め全 7 章から成る．次章では本研究の基盤環境であるホームコンピューティングとホームコンピューティングにおける入力インタフェースを論考する．続く第 3 章では，既存の身振りによる入力インタフェースの持つ問題点を指摘し，ついで本研究のアプローチを述べる．また，第 4 章ではシステムの概要を説明するとともに，システムの設計について述べる．第 5 章では，その実装について詳述し，第 6 章で本研究の評価を行う．第 7 章にて本論文をまとめ，今後の課題について言及する．

第 2 章

ホームコンピューティングにおける入力インタフェース

本章では，本研究が想定するホームコンピューティングにおける入力インタフェースについて述べる．まず，ホームコンピューティングを定義し，ホームコンピューティングの利用者について述べる．ついで，ホームコンピューティングの利用者の特徴からホームコンピューティングにおける入力インタフェースの要件を考察する．続いて，従来の入力インタフェースを再考し，その問題点について言及する．

2.1 ホームコンピューティング

ホームコンピューティングとは主に家庭を対象としたユビキタスコンピューティングを指す．ホームコンピューティングでは，コンピュータはテーブルトップでの利用を想定しない．このため，コンピュータは専用ディスプレイやキーボード，マウスを備えた従来のパーソナルコンピュータの形状ではなく，情報家電や家具，壁といったものに埋め込まれるのに適した形状で利用者の身の回りに遍在する．本論文では，これらのコンピュータのうち，操作可能なコンピュータを操作対象と呼ぶ．これらのコンピュータはネットワークで相互に通信しながら動作し，利用者の行動を支援する．このようなさまざまな形状コンピュータがシームレスにつながり，連携動作させるためのシステムソフトウェアも研究されている [14, 5, 11, 9]．また，このようなホームコンピューティングにおける様々な操作対象を操作するためのインタフェースはリモコンやディスプレイによる GUI が一般的であり，操作対象の多機能化によって操作系が複雑化する傾向にある．

以下に，ホームコンピューティングの特徴を整理する．

- 多種多様な形状の操作対象が存在する
- 操作対象の機能が高機能化，多機能化する
- 複数の操作対象が連携しながら動作する

2.1.1 ホームコンピューティングの利用者と特徴

ホームコンピューティングの利用者はコンピュータの専門家でなく，コンピュータの知識が乏しいことが想定される．このため，ホームコンピューティングではコンピュータに詳しくない利用者でも操作対象を簡単に扱えるようにする必要がある．

図 2.1　ながら作業

また，家庭における利用者の行動には「ながら作業」が多く見受けられる．例えば，歩きながら左手に新聞を持ち，新聞を読みながら右手でリモコンを持ち，リモコンでテレビを操作するという行動である．本論文では，このような利用者の並列的な連続した一貫作業を「ながら作業」と呼ぶ．ながら作業を図 2.1 に示す．以下に，ホームコンピューティングの利用者の特徴を整理する．

- コンピュータの知識に乏しい
- ながら作業をする

2.2　ホームコンピューティングにおける入力インタフェース

本節では第 2.1 節で述べたホームコンピューティングの利用者の特徴をもとに，ホームコンピューティングにおける入力インタフェースについて考察する．まず，ホームコンピューティングにおける入力インタフェースの要件をまとめ，ついで従来の入力インタフェースを整理する．

2.2.1　ホームコンピューティングにおける入力インタフェースの要件

ホームコンピューティングにおける入力インタフェースは，ホームコンピューティングの利用者に適応する必要がある．本項では，ホームコンピューティングにおける入力インタフェースが満たすべき要件を述べる．

直感性の高いインタラクション

コンピュータの知識に乏しい利用者が使える入力インタフェースを実現するためには，直感性の高いインタラクションを提供する必要がある．入力インタフェースの直感性には二つの直接感が重要になる [24]．一つ目

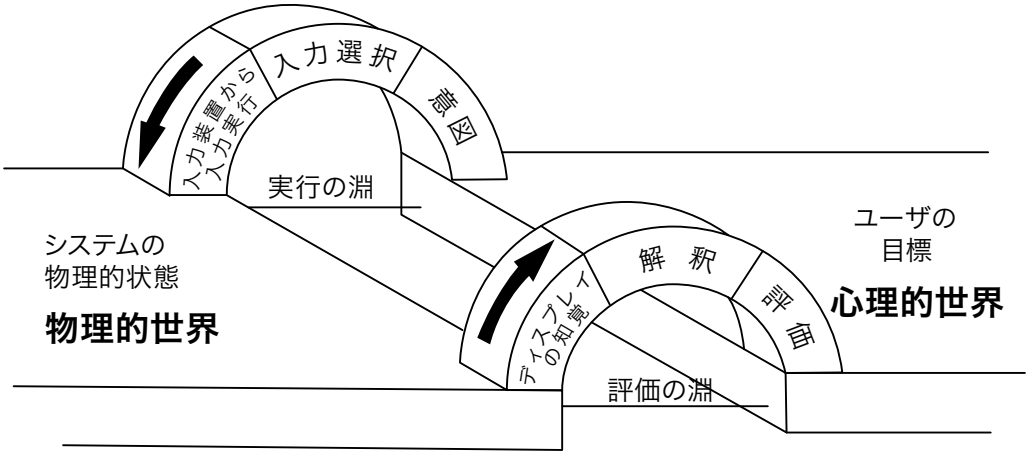


図 2.2　D.A.Norman の 7 段階行為モデル [21]

1. ゴールの形成：自分のゴールを設定する段階
2. 意図の形成：ゴールに対して，自分の取るべき行動を決定する段階
3. 行為の詳細化：適切な行動を決定する段階
4. 行為の実行：物理的世界に働きかける段階
5. 外界の状況の知覚：実行によって物理的世界の状態変化を知覚する段階
6. 外界の状況の解釈：知覚された情報の意味を解釈する段階
7. 結果の評価：自分が定めたゴールと比較し，ゴールの達成を評価する段階

図 2.3　D.A.Norman の 7 段階行為系列

が実行過程と評価過程の直接感で二つ目が対象操作の直接感である．実行過程と評価過程とは D.A.Norman が提案した 7 段階の行為モデル [10] における行為の遷移過程である．D.A.Norman による 7 段階の行為モデル図とその行為系列を図 2.2 と図 2.3 に示す．

D.A.Norman は人間が外界とのインタラクションを行う際に，この 7 段階の行為系列を経ると考え，2 つ目のゴールの形成から 4 つ目の行為の実行までの行為系列をを実行過程，5 つめの外界の状況の知覚から 7 つめの結果の評価までの行為系列を評価過程と定義した．図 2.2 においてはそれぞれ実行の淵，評価の淵と表現されている．つまり，実行過程と評価過程の直接間を高めることは図 2.2 における実行の淵，評価の淵の距離を縮めることである．

これら二つの行為系列の直接感を高めるためには現実のオブジェクトの情報や位置情報など，現実世界における人間の状況を利用する必要がある．現実世界のオブジェクトの情報を利用することで，利用者に直感性の高いインタラクションを提供できる．

身体情報の利用

ながら作業を支援するためには，身体情報を利用する必要がある．例えば，リモコンで片手がふさがれている状況で，両手による作業はできない．このような場合，利用者の視線や体の向き，手の動き，声などの利用

者の身体情報を利用して操作対象を操作する．これにより，操作よる物理的制限を軽減し，ながら作業を支援できる．

まとめ

直感性の高いインタラクションと身体情報の利用の 2 項目はホームコンピューティングにおける入力インタフェースが共通して備えるべき機能や性質である．特に，最後の項目はながら作業の支援という面だけでなく，インタラクションの直感性にもつながる項目である．

2.3 従来の入力インタフェース

今日の家庭には，コンピュータや家電機器を操作するために様々な入力インタフェースが存在する．本節では，まず，従来の入力インタフェースを分類し，ついで，第 2.2.1 項で整理したホームコンピューティングにおける入力インタフェースの要件をもとに考察する．

2.3.1 リモートコントローラ

現在，リモートコントローラ（以下，リモコン）は家庭でもっとも普及している入力インタフェースである．主に家電機器を遠隔から操作することにより，利用者の身体的負担を軽減することを目的とする．近年，リモコンでないと操作できない家電機器も登場し，ほぼすべての家電機器に専用リモコンが付属する．リモコンは赤外線インタフェースにより家電機器と通信することが一般的なため，これを利用して複数の家電機器をコントロールする学習型リモコン [20] や，携帯電話アプリケーションも開発されている．また，ディスプレイと GUI を利用することで，動的にリモコン画面を生成し，家電機器を一元的に管理するユニバーサルコントローラの研究開発も行われている [22].

リモコンやユニバーサルコントローラは，現実世界のオブジェクトの情報を利用していないため，直感性の高いインタラクションを提供しているとは言い難い．また，利用者の身体情報も利用していない．

2.3.2 Character-based User Interface（CUI）

Character-based User Interface（以下，CUI）は利用者に対する情報の提示を文字によって行い、すべての入力をキーボードを用いて行なう入力インタフェースである．Microsoft 社の MS-DOS，UNIX などのオペレーティングシステムで採用されており，現在でも，サーバ機器やネットワーク機器などを操作するための標準的な入力インタフェースである．図 2.5 に CUI の例を示す．

CUI は，現実世界のオブジェクトの情報を利用していないため，直感性の高いインタラクションを提供しているとは言い難い．また，利用者の身体情報も利用していない．

2.3.3 Graphical User Interface（GUI）

Graphical User Interface（以下，GUI）は利用者に対する情報の表示にグラフィックを利用し、大半の基礎的な操作をマウスなどのポインティングデバイスによって行なう入力インターフェースである．Microsoft 社の Windows，Apple 社の Mac OS などのパーソナルコンピュータ向けオペレーティングシステムをはじめとして，銀行の ATM や電車の券売機など様々な組み込み機器の入力インタフェースとして採用されている．

図 2.4　リモートコントローラ

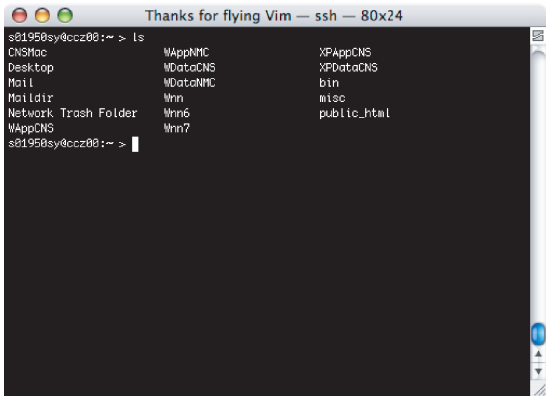


図 2.5　Character-based User Interface

グラフィックを利用することで，CUI より分かりやすい入力インタフェースを利用者に提供できる．従来型のリモコンが操作対象の機能に合わせて，ボタン式スイッチを用意する必要があるのに対し，GUI は操作対象の機能に合わせてグラフィックを変化させ，専用の入力インタフェースを生成できる．

また，マウスだけでなく，タッチディスプレイや装着型デバイスを利用して手の動きなどの身体情報を用いるポインティングデバイスも存在する．しかし，このような身体情報を用いるポインティングデバイスを用いても，ディスプレイのポイントされた座標のみが操作対象の制御変数である．つまり，手がどのように動いたかではなく，ディプレイ上のポインタがどのように動いたかが意味を持つ．このように，身体情報が直接的に操作対象の操作に影響しないという点で後述する身振りインタフェースとは性質が異なる．

GUI は，現実世界のオブジェクトの情報を利用していないため，直感性の高いインタラクションを提供しているとは言い難い．タッチディスプレイや装着型デバイスなどにより利用者の身体情報を利用しているが，十分でない．

2.3.4　Tangible User Interface（TUI）

Tangible User Interface [7]（以下，TUI）とは，従来はコンピュータのディスプレイで表現していた物理的実体を持たない情報に物理的表現を与え，直接触れて感知・操作可能にする．これにより，人間が活動する実世界とコンピュータが活動する仮想世界の間に存在する隔たりを埋め，直接操作性を高めようというアプローチである．ユニバーサルコントローラや GUI のように，汎用的なインタフェースを目指さず，特定のアプリケーションドメインにおける操作向上を目的に，特化されたインタフェースを目指している [6]．図 2.6 に TUI の例を示す．

TUI は，現実世界のオブジェクトの情報を利用しており，直感性の高いインタラクションを提供しているとはいえる．また，利用者の身体を利用しておらず，すべての操作は実世界のオブジェクトによって行われるため，ながら動作を支援しているとはいえない．

2.3.5　**身振りインタフェース**

身振りインタフェースは人間の身体の動きをセンサやカメラなどにより認識し，操作対象への入力を行うインタフェースである．人間の身体の動きとは，顔の向きや視線，手の動きなど様々である．身振りインタ

図 2.6　I/O Brush [13]

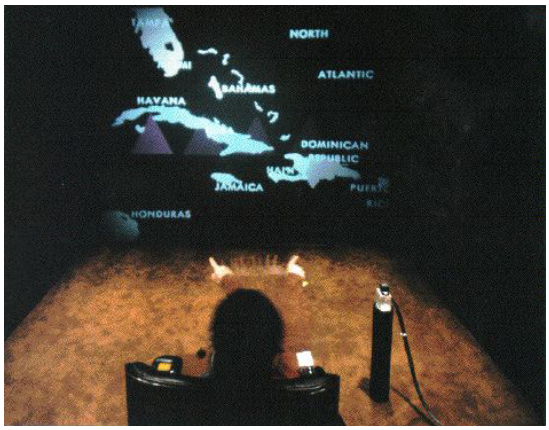


図 2.7　Put-That-There [2]

フェースでは身体の形状や動作パターンが操作に影響するのに対し，従来の入力インタフェースでは身体が動かす装置の制御変数が操作に影響する．例えば，前述の GUI のポインティングデバイスの例が挙げられる．また，車のハンドル操作では手がどのように動いたかではなく，ハンドルのどのように動いたか，つまり，ハンドルの回転角度によって車は操作されている．図 2.7 に身振りインタフェースの例を示す．

身振りインタフェースでは，操作対象を操作する手段が利用者の身体であるということから，現実世界のオブジェクトの情報を利用しており，直感的なインタラクションを提供できるといえる．

2.3.6　音声インタフェース

音声インタフェースはマイクを用いて利用者の音声を感知し，操作対象への入力を行うインタフェースである．同じ文字によるインタラクションを提供する CUI が定型のコマンドを定めているのに対し，音声インタフェースはより自然言語に近づけるアプローチがとられている．また，音声インタフェースでは音声の文字列だけではなく，音程や音量，調子など様々な情報が利用できる．

音声インタフェースは，現実世界のオブジェクトの情報を利用していないため，直感性の高いインタラクションを提供しているとは言い難い．また，利用者の身体情報も利用しない．

2.3.7　従来の入力インタフェースの比較

ここで，従来の入力インタフェースを前述のホームコンピューティングにおける入力インタフェースの要件をもとに比較する．諸要件の実現可能性を比較することにより，最もホームコンピューティングにおける入力インタフェースに適した入力インタフェースを考察する．この比較結果を表 2.1 に示す．

この表から，身振りインタフェースが最もホームコンピューティングにおける入力インタフェースに適していることが分かる．

表 2.1　ホームコンピューティングにおける入力インタフェースの分類と比較

入力インタフェースの要件	リモコン	CUI	GUI	TUI	身振り	音声
直感性の高いインタラクション	×	×	×	○	○	×
身体情報の利用	×	×	△	×	○	×

2.4　まとめ

本章では，本研究が扱うコンピューティング環境であるホームコンピューティングについて述べた．ホームコンピューティングでは多種多様な操作対象が存在し，それらが連携しながら動作する．また，ホームコンピューティングの利用者はコンピュータの知識に乏しく，主にながら作業を行う．

ついで，ホームコンピューティングにおける入力インタフェースの要件として，ホームコンピューティングの利用者の面から 2 つの指標を掲げた．この指標をもとに，従来の入力インタフェースをリモコン，CUI, GUI, TUI, 身振りインタフェース，音声インタフェースに分類し，それぞれを比較した．その結果，ホームコンピューティングにおける入力インタフェースとして身振りインタフェースが最も適していることを示した．

第 3 章

身振りを用いた入力インタフェース

本章では，身振りを用いた入力インタフェースについて述べる．まず，ホームコンピューティングにおける身振りインタフェースの要件を考察する．続いて，身振りインタフェースの先行研究を整理し，その問題点について言及する．これにより本研究の意義を確認した後，本研究のアプローチを述べる．

3.1 ホームコンピューティングにおける身振りインタフェース

本節では第 2.1 節で述べたホームコンピューティングの特徴をもとに，ホームコンピューティングにおける身振りインタフェースについて考察する．まず，ホームコンピューティングにおける身振りインタフェースの要件をまとめ，ついで身振りインタフェースの先行研究を整理する．

3.1.1 ホームコンピューティングにおける身振りインタフェースの要件

ホームコンピューティングにおける身振りインタフェースは，ホームコンピューティングが提供するアプリケーションに適応する必要がある．本項では，ホームコンピューティングにおける身振りインタフェースが満たすべき要件について述べる．

統一的な操作系

操作対象の機能が高機能化，多機能化するホームコンピューティングのなかで，多種多様な形状の操作対象に対応した入力インタフェースを提供するためには，統一的な操作系をもつ必要がある．現在，家電機器の操作に用いられるリモコンやボタンのように機能ごとに身振りを定義すると，各操作対象ごとに異なる操作系となり，それぞれ専用の入力インタフェースを利用者に提供することになる．利用者にとって，各操作対象ごとに操作方法が異なることは，操作方法を習得するうえで大きな負担となる．統一的な操作系を定義することで一つの入力インタフェースで多種多様な操作対象の機能を操作できる．

複数の操作対象間の接続指示

複数の操作対象を連係動作せるためには，複数の操作対象間の接続を指示する必要がある．ホームコンピューティングにおいては，操作対象はネットワークで相互に接続しながら，連係動作する．第 2.1 節で述べたとおり，さまざまな形状のコンピュータがシームレスにつながり，連携動作させるためのシステムソフトウェアも研究されている．しかし，利用者が望んでいる連携状態をシステムに伝えるためには，一般に IP アドレスや名前の指定などの煩雑な設定が必要となる．利用者が身振りで複数の操作対象間の接続指示を実現することで，操作対象の連係動作をより直接的に指示できる．

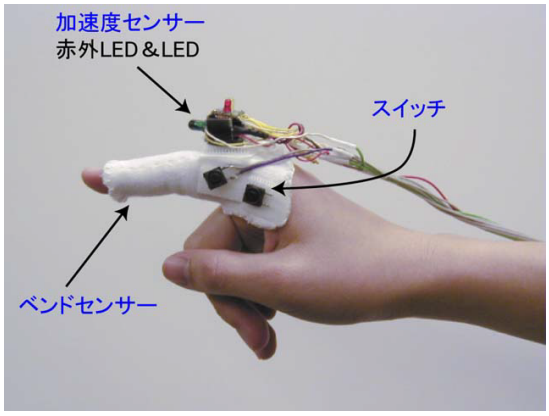


図 3.1　Ubi-Finger [16]

図 3.2　UbiButton [19]

統一的な操作系と複数の操作対象間の接続指示の 2 項目はホームコンピューティングにおける身振りインタフェースが共通して備えるべき機能や性質である．特に，統一的な操作系については，多種多様な操作対象に対応するだけでなく，操作対象が高機能化，多機能化することによって，複雑化する操作系を一般化，単純化することにより，利用者の操作にかかる負担の低減にもつながる項目である．

3.2　関連研究

本研究は，身振りの中でも手指の動きを入力インタフェースに利用する．本節では，手指を用いた入力インタフェースの関連研究を紹介する．

3.2.1　Ubi-Finger

Ubi-Finger [16] は慶應義塾大学，政策メディア研究科の塚田浩二氏が開発した装着型ジェスチャ認識デバイスである．Ubi-Finger は 3 系統のセンサ（ベンドセンサ，2 軸加速度センサ，タッチセンサ）を中心に，実世界の情報機器を特定するための赤外線トランスミッタと，これらのデバイスの制御やホスト PC などと通信を行うマイコンから構成される．図 3.1 に Ubi-Finger のプロトタイプを示す．

Ubi-Finger は，装着型デバイスによって認識されたジェスチャによって，自然なジェスチャによる携帯情報機器や情報家電機器の操作を実現する．Ubi-Finger を用いて機器を操作する場合，利用者は以下の手順をふむ．まず，利用者は操作対象の機器を指さし，赤外線トランスミッタにより ID そ送信することで，機器を「選択」する．次に，操作対象ごとに定められたジェスチャにより，操作対象を操作する．利用者がこれまで機器を操作していたメンタルモデルを利用してジェスチャ設計することで操作対象の機器が増加しても操作が複雑になることなく，既存の操作メタファや身体性を活用した直感的な操作を実現できる．

しかし，Ubi-Finger は操作対象の機器ごとに異なるジェスチャを設定しており，統一的な操作系をもたない．結果として，利用者は操作対象ごとにジェスチャを学習しなければならない．また，多種多様な操作対象が存在し，それらの操作対象が高機能化，多機能化すると想定されるホームコンピューティングにおいて，既存のメンタルモデルでは対応できない機器操作も想定されるため，利用者の学習コストを下げるアプローチとして不十分である．Ubi-Finger は赤外線トランスミッタを利用して機器を選択することが可能であるが，複

表 3.1　UbiButton の統一コマンド例 [19]

Command	PDA/Cursor	TV	VCR	AirCon	Room Light
On/Off	Exec.	On/Off	Play/Stop	On/Off	On/Off
Up	Up	Vol.Up	Vo.Up	Temp.Up	Dimm.Up
Down	Down	Vol.Dn	Vol.Dn	Temp.Dn	Dimm.Dn
Fwd	Right	Ch.Fwd	FF	-	-
Rev	Left	Ch.Rev	Rew	-	-

数の操作対象間の接続指示について考慮されていない．

3.2.2　UbiButton

UbiButton [19] は NTT ヒューマンインタフェース研究所の福本雅朗氏が開発した装着型ジェスチャ認識デバイスである．UbiButton は 1 軸の加速度センサと金属薄板によって構成される腕時計型デバイスである．この腕時計型デバイスを手首に装着することで，任意の指先で行ったタイピング動作（打指）の有無を検出する．また，親指と他の指どうしを軽く接触させる「指パチ」入力によって，机などの支持物体がない場合でも入力が可能である．図 3.2 に UbiButton を示す．

UbiButton は，装着型デバイスにより認識された指パチによって，情報家電機器を操作することを想定している．しかし，UbiButton は打つと打たない 1 ビットの情報しか入力できない 1 ビット入力装置であるため，複数のコマンドを入力するために時間情報を用いる必要がある．UbiButton では指パチと指パチされた時間間隔によってコマンドを表現する方式として MORSL 方式を提案している．しかし，実用性を考慮すると表現可能なコマンド数には限りがある．このため，統一的なコマンドを用意し，使用時に操作対象機器を指定することで，多数の機器操作に対応している．表 3.1 に UbiButton の統一コマンド例を示す．

しかし，UbiButton は統一されたコマンドにおいても，ひとつのコマンドを入力するために 2〜4 ストローク必要であり，コマンドの入力に時間がかかる．即時性が求められる機器操作や，連続して操作する場合に問題になる．また，UbiButton は操作対象を指定することができないため，複数の操作対象間の接続指示するための方式を別に利用する必要がある．

3.2.3　GestureWrist

GestureWrist [12] はソニー株式会社ソニーコンピュータサイエンス研究所インタラクションラボラトリーの暦本純一氏が開発した装着型ジェスチャ認識デバイスである．GestureWrist はピエゾアクチュエータと電極，加速度センサ，腕時計によって構成される腕時計型デバイスである．この腕時計型デバイスを手首に装着することで，手指の形状と前腕の動作を検出する．図 3.3 に GestureWrist を示す．

GestureWrist では機器制御について考慮されていない．また，複数の操作対象間の接続を指示するための方式も考慮されていない．

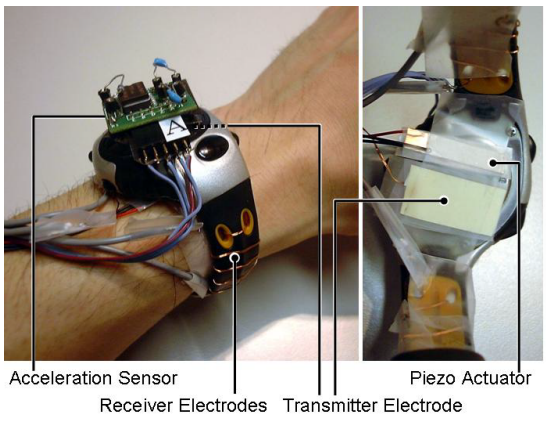


図 3.3　GestureWrist [12]

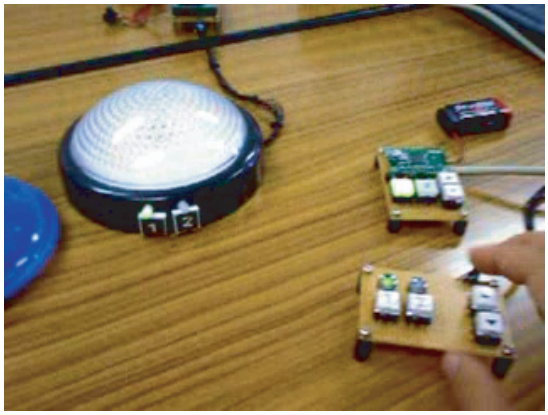


図 3.4　Touch-and-Connect [23]

3.2.4　Touch-and-Connect

Touch-and-Connect [23] は名古屋大学大学院情報科学研究科の岩崎陽平氏が開発したボタン型デバイスを用いたユビキタス環境における接続指示フレームワークである．Touch-and-Connect はボタン型デバイスと，情報提示インタフェース，機器管理サービス，TnC プロトコルサービス，メッセージングサービスから構成される．図 3.4 に Touch-and-Connect のボタン型デバイスを示す．

Touch-and-Connect はボタン型デバイスを各機器にとりつける．このボタン型デバイスはプラグボタン（P）とソケットボタン（S）の二つのボタンから構成される．機器 A と機器 B を接続する場合には，まず，機器 A のプラグボタンを押すことによってそれが接続元機器であることを指示し，その後，機器 B のソケットボタンを押すことによってそれが接続先機器であることを指示する．このように，Touch-and-Connect では連続的に機器にとりつけられたボタンを押すことで，二つの機器間の接続を指示し，二つの機器の連携動作を実現する．しかし，Touch-and-Connect は機器の操作インタフェースは持たず，機器操作については考慮されていない．

3.3　本研究の意義

本節では，第 3.2.1 項で述べたホームコンピューティングにおける身振りインタフェースの要件をもとに，関連研究を比較，考察する．関連研究を諸要件について比較することにより，関連研究の問題点を指摘し，本研究の意義を示す．

3.3.1　関連研究の比較

ホームコンピューティングにおける身振りインタフェースの要件は，統一的な操作系と複数の操作対象間の接続指示の 2 つであった．これらの要件を軸とした比較結果を表 3.2 に示す．

この表が示すとおり，関連研究ではホームコンピューティングにおける身振りインタフェースの要件を満たさないため，ホームコンピューティングにおける身振りインタフェースを構築することは困難である．2 つの要件を満たす身振りインタフェースが必要である．

表 3.2　関連研究の比較

身振りインタフェースの要件	UbiFinger	UbiButton	GestureWrist	Touch-and-Connect
統一的な操作系	×	○	×	×
複数の操作対象間の接続指示	△	×	×	○

3.4　本研究のアプローチ

本節では，本研究がホームコンピューティングにおける身振りインタフェースを開発するに当たってのアプローチを述べる．

3.4.1　統一的な操作系

統一的な操作系を実現するためには，多種多様な操作対象が持つ様々な機能と動作を抽象化し，整理する必要がある．機能と動作を抽象化することにより，限られた命令のみで多種多様な操作対象を操作できる．本研究では，利用者の学習にかかる負担を軽減するために，機能に適した操作系を設計するのではなく，より少ない命令のみで構成される操作系を設計するアプローチをとる．

ホームコンピューティングにおける操作対象の動作は，操作対象が持ついくつかの内部変数の変化として捉えることができる．例えば，テレビはチャンネルと音量，電源などを内部変数として持ち，利用者が操作するリモコンからの入力により，これらの内部変数を変化させる．これにより，テレビは状態を変え，利用者から操作される．すなわち，ホームコンピューティングにおける操作対象は以下に示す 3 つの命令だけで操作できる．

1. 操作対象の選択
2. 遷移させる内部変数の選択
3. 内部変数の遷移を指示

本研究ではこれらの 3 つの命令を発行する手振りインタフェースを開発する．これにより，多種多様な操作対象が持つ様々な機能を統一的な操作系で操作できる．

3.4.2　複数の操作対象間の接続指示

ホームコンピューティングにおける操作対象間の接続は，接続元操作対象と接続先操作対象を指定することによって実現される．すなわち，ホームコンピューティングにおいて複数の操作対象へ接続指示するためには，以下に示す 2 つの命令が必要である．

1. 接続元操作対象の指定
2. 接続先操作対象の指定

本研究ではこれらの 2 つの命令を発行する手振りインタフェースを開発する．これにより，複数の操作対象間の接続を指示し，複数の操作対象を連係動作させることができる．

表 3.3　SUI の統一コマンドと意味

コマンド	意味
Nomination	操作対象の選択
Change	遷移させる内部変数の変更
Next/Previous	内部変数の遷移指示
Source	接続元操作対象の指定
Start/End	内部変数の遷移開始/内部変数の遷移終了

3.5　身振りの設計

本節では，第 3.4 節で述べたアプローチに従い，身振りの設計について述べる．まず，本研究で開発するコマンド体系について述べる．ついで，コマンドと身振りのマッピングについて述べる．

3.5.1　コマンド体系

第 3.4 節では，操作対象の機能を操作するためには，3 つの命令が必要なことと複数の操作対象間の接続指示のために，2 つの命令が必要なことを述べた．これらの命令をシステムが受け付けるために，Nomination，Change，Next/Previous，Source，Start/End から構成される統一コマンドを設計した．表 3.3 に統一コマンドを示す．それぞれのコマンドについて，以下で詳述する．

Nomination

Nomination コマンドは操作対象を指定した際に，操作対象へ送られるコマンドである．操作対象は Nomination コマンドを受け取ることで，自身が利用者に操作対象として選択されたことを知る．Nomination コマンドを受け取った操作対象は，利用者に対して自らが選択されたことを示すために様々な方法でフィードバックする．これにより，利用者は環境に数多く存在する操作対象の中から，利用者が意図した操作対象が正しく選択されていることを確認できる．

Change

Change コマンドは操作対象を指定した後，利用者が遷移させたい内部変数を変更した際に，操作対象へ送られるコマンドである．例えば，テレビがチャンネルと音量という 2 つの内部変数を持つ場合，Change コマンドが受け取ることで，遷移させるアクティブな内部変数を変更する．もし，チャンネル変数がアクティブなときに Change コマンドを受け取ると，操作対象は音量変数へとアクティブな内部変数を変更する．利用者は Change コマンドにより遷移させたい内部変数を選択し，後述する Next/Previous コマンドで内部変数を遷移させる．

Next/Previous

Next/Previous コマンドは操作対象の内部変数を遷移させる相対的な変数を指定するためのコマンドである．一般に，内部変数を遷移させる方法として，相対値指定と絶対値指定がある．例えば，テレビのチャンネルを 2 から 8 に変更する場合，二通りの方法で変更できる．一つめが，リモコンの 8 チャンネルボタンを押し，8 という絶対値をテレビに送信する方法である．二つめがリモコンのチャンネルを

表 3.4　コマンド体系に基づく操作例

Command	TV	AirCon	Room Light
Nomination	操作対象の指定	操作対象の指定	操作対象の指定
Change	チャンネル/音量	温度/風量	-（光量）
Next	次のチャンネル/音量大	温度上/風量大	光量大
Previous	前のチャンネル/音量小	温度下/風量小	光量小
Source	接続元への接続要求	接続元への接続要求	接続元への接続要求
Start	チャンネル/音量変更の開始	温度/風量変更の開始	光量変更の開始
End	チャンネル/音量変更の終了	温度/風量変更の終了	光量変更の終了

一つ増加させるボタンを 6 回押し，現在のチャンネルから数えて 6 という相対値をテレビに送信する方法である．前者の絶対値を用いる方法では，1 回の入力でチャンネルを変更できるのに対し，後者の相対値を用いる方法では，6 回の入力が必要である．しかし，n 個の絶対値が存在する場合前者の方法を実現するためには n 個の絶対値入力コマンドが必要になる．本研究では，より少ないコマンドのみで構成される操作系を目指すため，絶対値指定は適さない．このため，Next/Previous という 2 つの相対値指定コマンドのみで，内部変数の遷移を実現する．

Source

Source コマンドは接続先の操作対象に対して送られるコマンドである．接続先の操作対象は Soure コマンドを受け取ることで，接続元の操作対象を知る．Source コマンドを受け取った操作対象は，接続元の操作対象と通信し，連携的に動作する．

Start/End

Start/End コマンドは，Next/Previous コマンドを送る身振りのはじめとおわりを示すためのコマンドである．Start/End コマンドを受け取った操作対象は，利用者に対して自らが Next/Previous コマンドを受け付ける状態であることを示すために様々な方法でフィードバックする．これにより，利用者は操作対象が操作の開始，終了を正しく認識していることを確認できる．

本研究におけるコマンド体系はこれらのコマンドから構成される．このコマンド体系に基づく操作例を表 3.4 に示す．

3.5.2　コマンドと身振りのマッピング

第 3.5.1 項では 7 つのコマンドから構成される統一コマンドについて述べた．これらのコマンドに対応する手指による身振りとして，さす，うつ，つまむ，はなす，まわすという 5 つの身振りを用いる．図 3.5 にこれらの身振りを示す．また，これらの身振りと統一コマンドのマッピングを表 3.5 に整理した．それぞれの身振りとコマンドのマッピングについて，以下で詳述する．

さす

さすという身振りは，操作対象を指さす動作である．さす身振りは Nomination コマンドとマッピングされる．利用者は操作したい操作対象を指さすことで，操作対象を選択できる．利用者に指さされた操

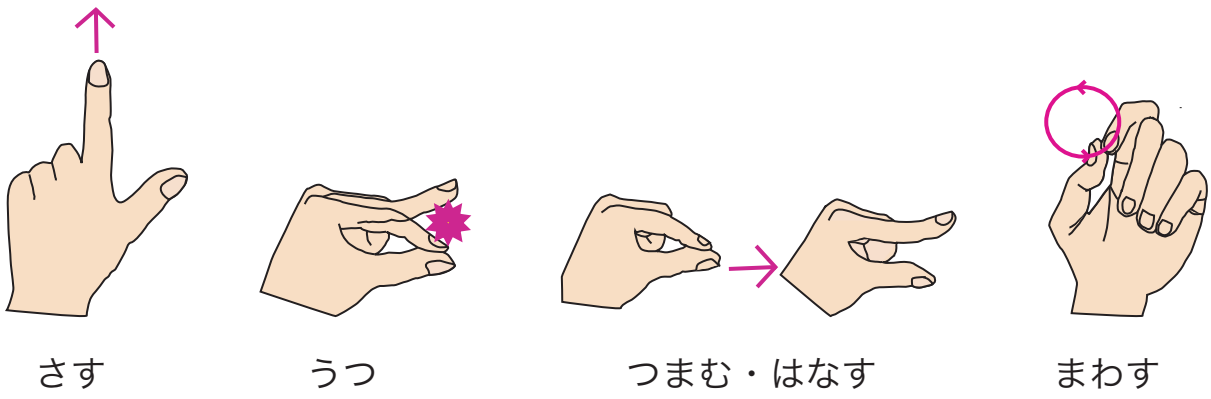


図 3.5　5 つの身振り

表 3.5　統一コマンドと身振りのマッピング

コマンド	身振り
Nomination	さす
Change	うつ
Next/Previous	まわす（右回り/左回り）
Source	（接続元をつまんだ後に）はなす
Start/End	つまむ/はなす

作対象へは Nomination コマンドが送られる．

うつ

うつという身振りは，親指と人差し指の指先どうしを軽く接触させる動作である．うつ身振りは Change コマンドとマッピングされる．利用者は操作対象に向けて親指と人差し指の指先をうつことで，操作対象の遷移させたい内部変数を選択できる．利用者が指先をうつと，操作対象へは Change コマンドが送られる．

まわす

まわすという身振りは，後述するつまんだ指先で円を描くようにまわす動作である．まわす身振りは右回りと左回りの 2 通りある．右回りのまわす身振りは Next コマンドにマッピングされ，左回りは Previous コマンドにマッピングされる．利用者は操作対象に向けてつまんだ指先で円を描くことで，選択した内部変数を遷移させることができる．利用者からつまんだ指先をまわされた操作対象へは Next, Previous コマンドが送られる．

つまむ・はなす

つまむという身振りは，親指と人差し指の指先どうしを軽く接触させた状態を保つ動作で，はなすという身振りは，つまんだ指先を離す動作である．つまむ身振りは Start コマンドに，はなす身振りは End コマンドにマッピングされる．利用者は操作対象に向けて指先をつまむことで，内部変数操作の開始し，操作対象に向けて指先をはなすことで内部変数操作の終了する．利用者からつままれた操作対象には Start コマンドが送られ，はなされた操作対象には End コマンドが送られる．

3.6 まとめ

本章では，ホームコンピューティングにおける身振りインタフェースの要件として，ホームコンピューティングの特徴から 2 つの指標を掲げた．この指標をもとに，既存の身振りインタフェースである Ubi-Finger，UbiButton，GestureWrist，Touch-and-Connect についてそれぞれ考察を加えた．その結果，既存の身振りインタフェースでは，すべての指標を満たすことはできず，ホームコンピューティングにおける身振りインタフェースを構築することが困難であることを示した．

ついで，本研究がホームコンピューティングにおける身振りインタフェースを実現するためのアプローチについて述べた．このアプローチに従い，コマンド体系の設計し，コマンドと身振りのマッピングについて述べた．

第 4 章

SUI Framework の設計

本章では，ホームコンピューティングに適したアプリケーションフレームワークとして，SUI Framework を設計する．SUI Framework は操作対象を操作するための入力インタフェースである SUI Controller モジュール，操作対象となる SUI Service モジュール，SUI Service の通信機能を担う SUI Service Container モジュール，SUI Service モジュールの情報を管理する SUI Directory Service モジュールによって構築される．はじめに，SUI Framework の設計方針について触れ，SUI Framework の概要と構成，動作概要について述べる．その後，SUI Framework を構成するモジュール群の設計について述べる．

4.1 SUI Framework の設計方針

SUI Framework の目的は統一的な操作体系に対応した入力インタフェースとグアプリケーションの作成を支援し，容易にホームコンピューティングを実現することである．本節では，の目的を達成するための指標を 3 つ掲げ，設計方針とする．それぞれ操作対象間の非依存性，フレームワークモジュール間の非依存性，簡便性である．

操作対象間の非依存性

操作対象 A と B が連携動作可能な場合，たとえ，操作対象 A が実行されていなくとも，もう片方の操作対象 B は単独で動作する必要がある．SUI Framework では，操作対象ごとに独立した通信機能を持たせ，操作対象間の依存性を排除する．これにより，システム管理者の管理コストを抑え，障害時においても利用者は利用可能な操作対象の機能を最大限利用できる．

フレームワークモジュール間の非依存性

SUI Framework を構成するモジュール間においてメッセージングプロトコルを定め，モジュール間の依存性を排除する．これにより，新たなモジュールの追加開発コストを低減するとともに，モジュールのどれかに障害が起きた場合においても，障害を最小限に抑えることができる．

簡便性

ホームコンピューティングでは，システムの利用時，開発時および導入時において，利用者，開発者，管理者それぞれの負担の軽減が求められる．したがって，SUI Framework では，導入や管理，開発コストを最小限に抑えることを目指す．

4.2 SUI Framework の設計概要

本節では，SUI Framework の構成として，ソフトウェア，ハードウェアの双方について述べる．ソフトウェア構成では，前述の設計方針に従って機能について述べる．ハードウェア構成では，必要となるコンピュータや機器の役割の構成について述べる．

4.2.1 ソフトウェア構成

本項では，ソフトウェア構成の概要と各モジュールの機能について述べる．SUI Framework のソフトウェアは SUI Controller モジュール，SUI Service モジュール，SUI Service Container モジュール，SUI Directory Service モジュールの 4 つから構成される．図 4.1 に SUI Framework のソフトウェア構成図を示す．以下でそれぞれについて解説する．

SUI Controller モジュール

SUI Controller モジュールとは利用者に入力インタフェースを提供し，利用者からの入力を受け付ける機能を持つモジュールである．SUI Controller モジュールは，利用者へ入力インタフェースを提供するコントローラ部から入力を受け，コマンド生成部で操作コマンドに変換する．この操作コマンドををコマンド送信部が SUI Service モジュールに操作コマンドを送信することで，操作対象である SUI Service モジュールを操作する．SUI Controller モジュールは，コントローラ部を入れ替えることで，様々な入力インタフェースを利用者に提供できる．本研究では，利用者の身振りを認識するコントローラ部を実装し，利用者の身振りにより操作対象を操作できる入力インタフェースを実現する．

SUI Service モジュール

SUI Service モジュールとは利用者から操作される操作対象である．SUI Service モジュールは，後述する SUI Service Container モジュールとアプリケーション部から構成され，SUI Service Container モジュールからアプリケーション部の内部変数を遷移させられることで動作する．SUI Service モジュールはアプリケーション部を入れ替えることで，SUI Framework に対応した操作対象を容易に構築できる．

SUI Service Container モジュール

SUI Service Container モジュールは，SUI Service モジュールの内部モジュールとして通信機能とコマンド実行機能を持つ．SUI Service Container モジュールは，通信機能を持つコマンド受信部が SUI Controller モジュールから操作コマンドを受け付け，コマンドキューにいれる．コマンド実行部は，コマンドキューにいれられた操作コマンドを逐次解析し，操作コマンドに合わせてアプリケーション部の機能を実行する．SUI Service Container はすべての SUI Service が内部に持つモジュールである．

SUI Directory Service モジュール

SUI Directory Service モジュールとは，SUI Service モジュールの情報を保持する機能を持つモジュールである．SUI Service モジュールの情報には，SUI Service モジュールの名前やそのモジュールへの参照情報などがある．SUI Directory Service モジュールは，SUI Framework が動作するネットワーク上に一つ以上存在し，SUI Service モジュールは起動時に自らの情報を登録する．また，SUI Controller モジュールは，操作コマンドを送信する際に，SUI Directory Service モジュールを参照する．

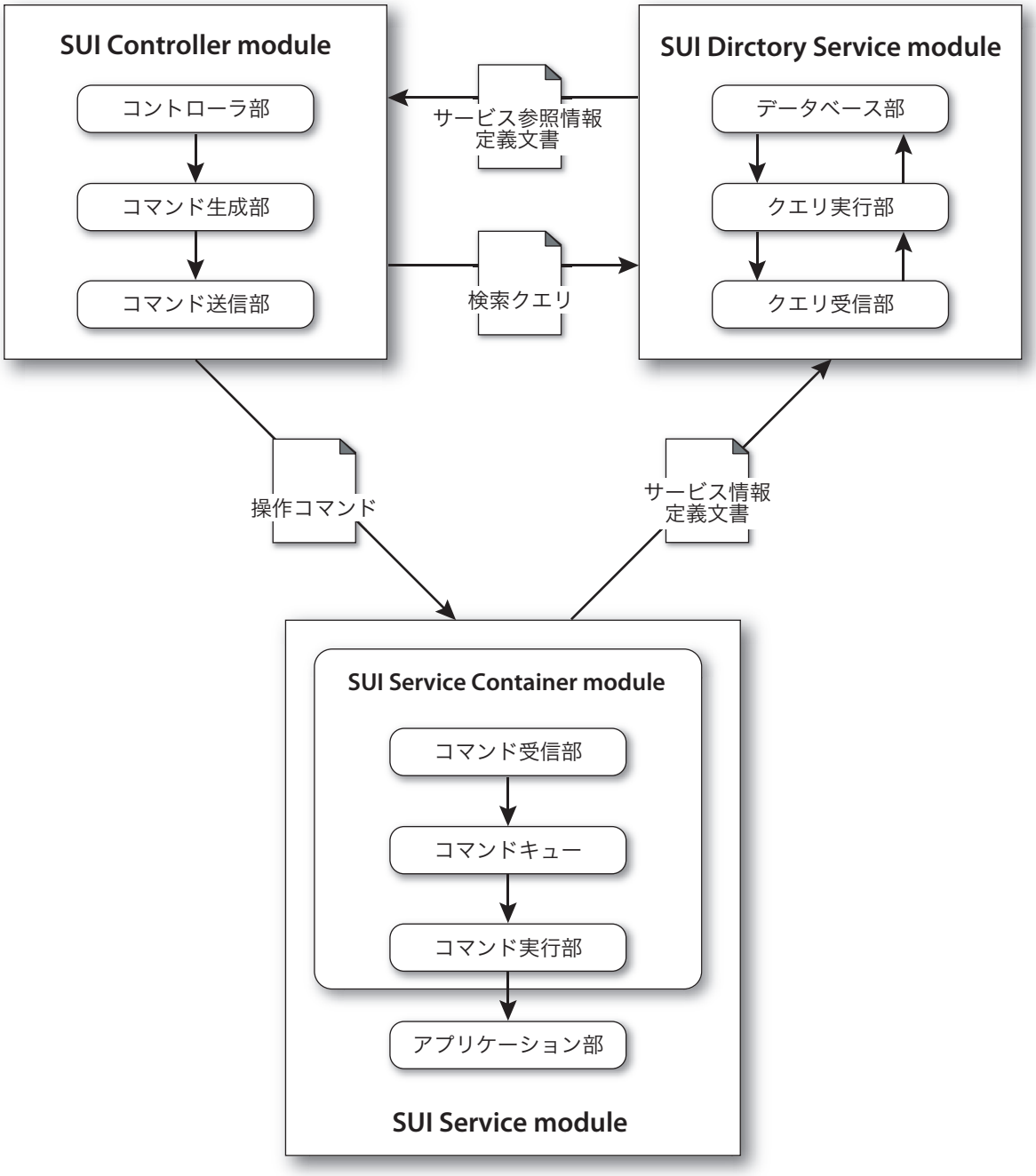


図 4.1　SUI Framework のソフトウェア構成図

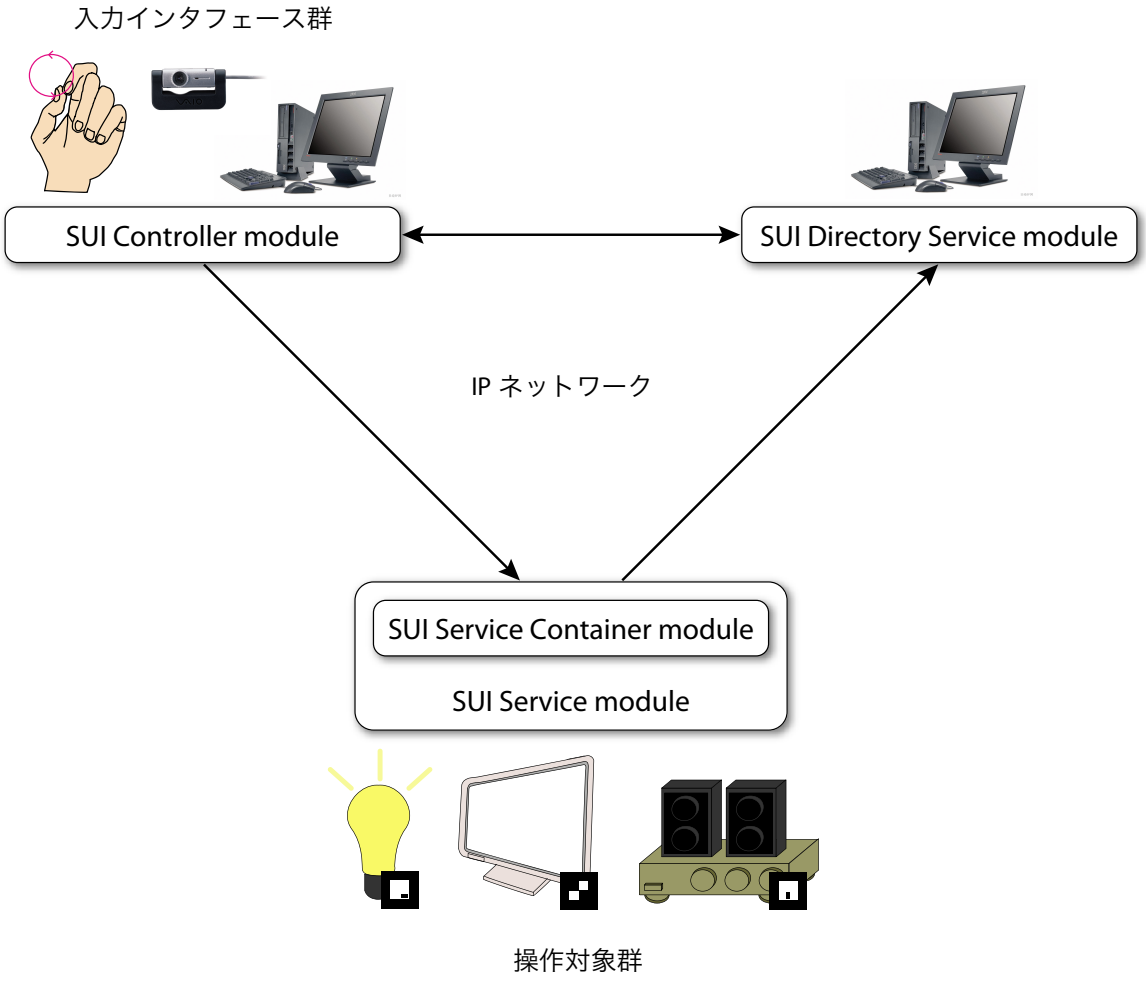


図 4.2　ハードウェア構成図

4.2.2　ハードウェア構成

SUI Framework でのハードウェア構成は，図 4.2 に示すように SUI Service モジュール，SUI Controller モジュール，SUI Directory Service モジュール，SUI Service Container モジュールから成る．SUI Controller モジュールと SUI Service モジュールはそれぞれホームネットワーク上に複数存在する．SUI Directory Service モジュールも複数モジュールが同時に動作可能であるが，ここでは，ホームネットワーク上に一つだけ動作していることを想定している．SUI Controller モジュール，SUI Directory Service モジュール，および SUI Service モジュール間の通信は IP ネットワークを介して行う．各モジュールは互いに依存性がなく，独立して動作する．

SUI Service モジュールは起動時に自らの情報を Directory Service モジュールに登録し，Directory Service モジュールはすべての SUI Service モジュール情報を管理する．SUI Controller モジュールは利用者からの操作要求を受け，Directory Service モジュールから操作対象となる SUI Service モジュールの情報を取得し，SUI Service モジュールへ操作コマンドを送る．これにより操作対象は利用者から操作される．

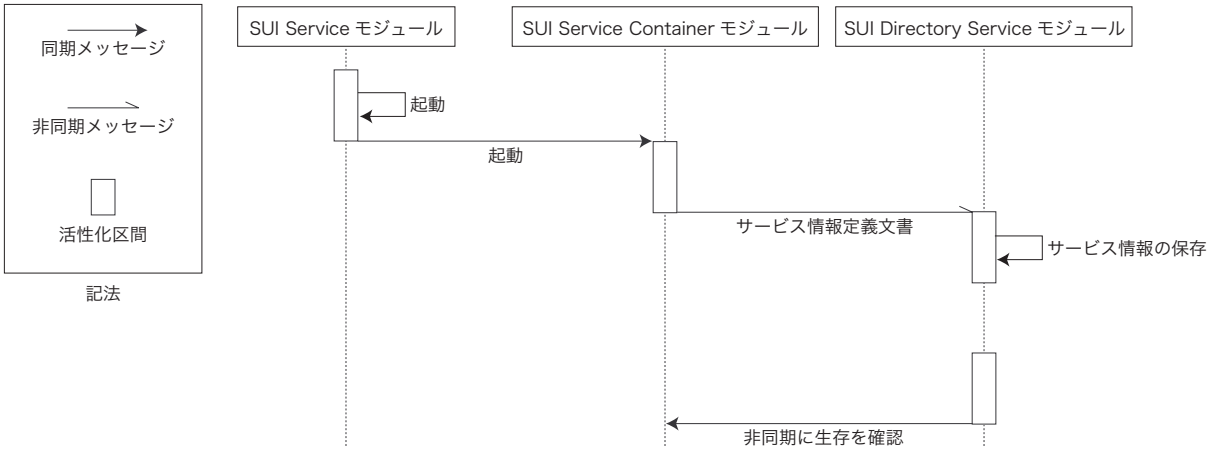


図 4.3　SUI Service の起動時の動作概要

4.3　SUI Framework の動作概要

本節では，SUI Framework の動作について説明する．SUI Framework の動作は大きく 2 つの動作に分けられる．一つめが SUI Service モジュールの起動から SUI Directory Service モジュールへ SUI Service 情報を登録するまでの一連の動作である．二つめが SUI Controller モジュールが利用者の入力を受けてから SUI Service モジュールが動作するまでの一連の動作である．いかにそれぞれの動作概要を述べる．なお，シーケンス図で利用する記法は Unfied Modeling Language [4]（以下，UML）を利用する．

4.3.1　SUI Service の起動時の動作概要

SUI Service モジュールの起動から SUI Directory Service モジュールへ SUI Service の情報を登録するまでの一連の流れを図 4.3 に示す．SUI Service モジュールは起動した際に，SUI Service Container モジュールを起動させる．起動した SUI Service Container モジュールは SUI Directory Service モジュールに対して，SUI Service の名前や参照情報などを定義したサービス情報定義文書を送信する．SUI Directory Service モジュールはサービス情報定義文書を受信し，データベース部に登録する．SUI Directory Service モジュールは，データベース部に登録されたサービス情報をもとに定期的に SUI Service Container モジュールの生存を確認する．

4.3.2　SUI Controller の動作概要

SUI Controller モジュールが利用者の入力を受けてから SUI Service モジュールが動作するまでの一連の流れを図 4.4 に示す．SUI Controller は利用者からの入力を受け，これをもとに操作コマンドを生成する．ついで，SUI Directory Service モジュールに対して SUI Service の情報を問い合わせるための検索クエリを送信する．SUI Directory Service モジュールはこの検索クエリをもとにサービス情報を検索し，生成したサービス情報定義文書を SUI Container モジュールへ返信する．SUI Controller モジュールは受信したサービス情報定義文書のサービス参照情報をもとに，SUI Service Container モジュールへ操作コマンドを送信する．

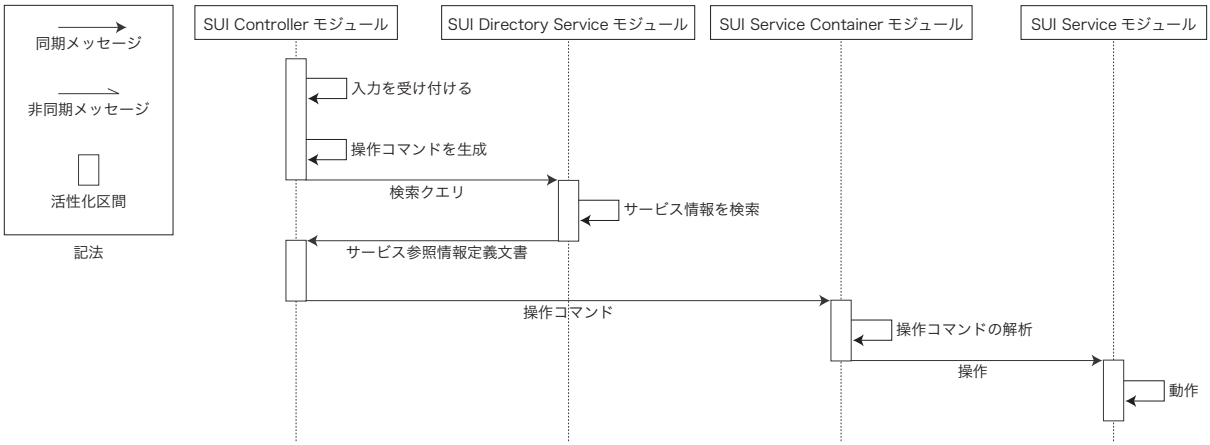


図 4.4　SUI Controller の動作概要

SUI Service Container モジュールは受信した操作コマンドを解析し，操作コマンドに合わせて SUI Service モジュールを動作させる．

4.4　SUI Controller モジュールの設計

SUI Controller モジュールは，コントローラ部，コマンド生成部，およびコマンド送信部から構成される．本項では，SUI Controller モジュールを構成するソフトウェア群の機能と SUI Service モジュールへ送信される操作コマンド，SUI Directory Service モジュールへ送信される検索クエリについて解説する．

4.4.1　コントローラ部

コントローラ部は利用者へ操作対象を操作するための入力インタフェースを提供する機能を持つ．本研究では，利用者の身振りを認識するコントローラ部を実装し，利用者の身振りにより，入力を受け付ける．コントローラ部は入れ替え可能であり，身振りを認識する機能だけではなく，GUI による入力インタフェースや，音声認識による入力インタフェースも SUI Framework 上で容易に開発できる．コントローラ部は，利用者の入力により，コマンド生成部で操作コマンドを生成し，コマンド送信部から SUI Service モジュールへ操作コマンドを送信する．

4.4.2　コマンド生成部

コマンド生成部はコントローラ部からの入力情報をもとに，第 3.5.1 項で述べた統一コマンドを生成する機能を持つ．操作コマンドについては第 4.4.4 項で解説する．コマンド生成部はコマンドを生成するための機能

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sui[
<!ELEMENT sui (command,ip?,port?)>
<!ELEMENT command (#PCDATA)>
<!ELEMENT ip (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ATTLIST sui command (manipulation) #REQUIRED>
]>
```

図 4.5　操作コマンド DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<sui command="manipulation">
    <command>source</command>
    <ip>127.0.0.1</ip>
    <port>20000</port>
</sui>
```

図 4.6　操作コマンド定義例

をコントローラ部へ提供する．

4.4.3　コマンド送信部

コマンド送信部はコマンド生成部で生成された操作コマンドをコントローラ部からの入力情報をもとに SUI Service モジュールへ送信する機能を持つ．コマンド送信部は操作コマンドを送信する前に，SUI Service モジュールのサービス参照情報を取得するため，SUI Directory Service モジュールへ検索クエリを送信する．検索クエリについては第 4.4.5 項で解説する．SUI Service モジュールのサービス参照情報を取得したコマンド送信部は，操作コマンドを SUI Service モジュールへ送信する．コマンド送信部はコマンドを送信するための機能をコントローラ部へ提供する．

4.4.4　操作コマンド

操作コマンドは，第 3.5.1 項で述べた統一コマンドを XML [3] で定義した XML 文書である．図 4.5 および図 4.6 に操作コマンドの DTD，および操作コマンドの定義例を示す．

SUI Framework における操作コマンドはすべてここに示した DTD に従い定義される．sui 要素の command 属性には manipulation を記述する．command 要素には，表 3.3 に示したコマンドのいずれかを記述し，ip 要素と port 要素は source コマンドの場合のみ，接続元 SUI Service モジュールへの参照情報として記述する．

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sui[
<!ELEMENT sui (id)>
<!ELEMENT id (#PCDATA)>
<!ATTLIST sui command (manipulation) #REQUIRED>
]>
```

図 4.7　検索クエリ DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<sui command="seeking">
    <id>e7fesb91-8cd5-0310-98dd-2f12e793c5e8</id>
</sui>
```

図 4.8　検索クエリ定義例

4.4.5　検索クエリ

検索クエリは，コントローラ部が取得した操作対象のネットワークにおける参照情報を取得するため，Directory Service モジュールへ送信される XML 文書である．図 4.7 および図 4.8 に検索クエリの DTD，および検索クエリの定義例を示す．

検索クエリは操作コマンドと同様に，sui 要素がルート要素となる．sui 要素の command 属性には seeking を記述する．id 要素には検索したい SUI Service の ID を記述する．

4.5　SUI Service モジュールの設計

SUI Service モジュールは，SUI Service Container モジュールとアプリケーション部から構成される．SUI Service Container モジュールは，コマンド受信部，コマンドキュー，コマンド実行部から成り，SUI Service モジュールの通信機能を担う．本項では，SUI Service モジュールを構成するソフトウェア群の機能と SUI Database Service へ送信されるサービス情報定義文書について解説する．

4.5.1　SUI Service Container モジュール

SUI Service Container モジュールは, SUI Service モジュールに内包されるモジュールである．SUI Service Container モジュールは SUI Controller モジュールから操作コマンドを受信し，操作コマンドを処理する．SUI Framework では，SUI Service Container モジュールを提供することで，SUI Service モジュールの開発コストを低減する．

4.5.2 コマンド受信部

コマンド受信部は SUI Service モジュールから第 4.4.4 項で定義した操作コマンドを受信する機能を持つ．コマンド受信部は，受信した操作コマンドを解析し，コマンドキューに格納する．

4.5.3 コマンドキュー

コマンドキューは操作コマンドを逐次実行するために，操作コマンドを順番に保持する機能を持つ．コマンドキューはコマンド受信部から操作コマンドを受け取り，順番毎に保持し，順番毎にコマンド実行部にわたす．コマンドキューはコマンド受信部へコマンドを格納するための機能を提供し，コマンド実行部へはコマンドを取得するための機能を提供する．

4.5.4 コマンド実行部

コマンド実行部は操作コマンドを実行する機能を持つ．コマンド実行部はコマンドキューから操作コマンドを受け取り，操作コマンドに合わせて後述するアプリケーション部の機能を操作する．

4.5.5 アプリケーション部

アプリケーション部は利用者から操作される内部変数を持ち，利用者にさまざまなフィードバックを返す機能を持つ．すなわち，第 1.2 節で述べたホームコンピューティングにおける操作可能なコンピュータであり，テレビや電灯などの機器制御，およびその他の様々な機能をもつソフトウェアコンポーネントである．アプリケーション部は，SUI Service Container と共に，SUI Service を構成し，SUI Controller から操作される．アプリケーション部はコマンド実行部に対し，内部変数を操作するための機能を提供する．

4.5.6 サービス情報定義文書

サービス情報定義文書は，サービス情報を登録するために，SUI Service の起動時に SUI Directory Service モジュールへ送信される XML 文書である．図 4.9 および図 4.10 にサービス情報定義文書の DTD，およびサービス情報定義文書例を示す．

サービス情報定義文書は sui 要素がルート要素となる．sui 要素の command 属性には registration を記述する．id 要素には自身の ID を記述し，name 要素には自身の名前を記述する．ip 要素と port 要素は，第 4.4.4 項で定義した操作コマンドのうち，source コマンドに記述される参照情報となる．

4.6 SUI Directory Service モジュールの設計

SUI Service モジュールは，SUI Service Container モジュールとアプリケーション部から構成される．SUI Service Container モジュールは，コマンド受信部，コマンドキュー，コマンド実行部から成り，SUI Service モジュールの通信機能を担う．本項では，SUI Service モジュールを構成するソフトウェア群の機能と SUI Database Service へ送信されるサービス情報定義文書について解説する．

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sui[
<!ELEMENT sui (id,name,ip,port)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT ip (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ATTLIST sui command (registration) #REQUIRED>
]>
```

図 4.9　サービス情報定義文書 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<sui command="registration">
    <id>e7fesb91-8cd5-0310-98dd-2f12e793c5e8</id>
    <name>Taro's PhotoFrame</name>
    <ip>127.0.0.1</ip>
    <port>20001</port>
</sui>
```

図 4.10　サービス情報定義文書例

4.6.1　クエリ受信部

クエリ受信部は，SUI Controller モジュールと SUI Service モジュールから，検索クエリやサービス情報定義文書を受信する機能を持つ．クエリ受信部は受信したメッセージをクエリ実行部に渡す．検索クエリを受信した場合は，クエリ実行部から検索結果を取得し，SUI Controller モジュールへ返信する．

4.6.2　クエリ実行部

クエリ実行部は，クエリ受信部からサービス情報定義文書や検索クエリを受け取り，これに合わせてデータベース部に操作を加える機能を持つ．サービス情報定義文書検索を受け取った場合には，データベース部に格納する．検索クエリを受け取った場合には，データベース部に格納されているサービス情報を検索し，検索結果をクエリ受信部に渡す．

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sui[
<!ELEMENT sui (id,name,ip,port)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT ip (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ATTLIST sui command (detection) #REQUIRED>
]>
```

図 4.11　サービス参照情報定義文書 DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<sui command="detection">
    <id>e7fesb91-8cd5-0310-98dd-2f12e793c5e8</id>
    <name>Taro's PhotoFrame</name>
    <ip>127.0.0.1</ip>
    <port>20001</port>
</sui>
```

図 4.12　サービス三章情報定義文書例

4.6.3　データベース部

データベース部はサービス情報を保持し，検索する機能を持つ．データベース部はクエリ実行部に対し，サービス情報を格納するための機能と，保持するサービス情報を検索する機能を提供する．

4.6.4　サービス参照情報定義文書

サービス参照情報定義文書は，SUI Controller モジュールからの検索クエリを受けて，SUI Controller へ返信される XML 文章である．SUI Service を参照するための情報が定義される．図 4.11 および図 4.12 にサービス参照情報定義文書の DTD，およびサービス参照情報定義文書例を示す．

サービス参照情報定義文書は sui 要素がルート要素となる．sui 要素の command 属性には detection を記述する．id 要素には検索クエリに記述されたされた SUI Service モジュールの ID を記述し，name 要素には SUI Service モジュールの名前を記述する．ip 要素と port 要素は，第 4.5.6 項で定義したサービス情報定義文書の ip 要素と port 要素が記述される..

4.7 まとめ

本章では，ホームコンピューティングに適したアプリケーションフレームワークである SUI Framework を設計した．SUI Framework の設計方針として，操作対象間の非依存性，フレームワークモジュール間の非依存性，簡便性を実現することを述べた．ついで，SUI Framework を構成するモジュールとして，操作対象を操作するための入力インタフェースを提供する SUI Controller モジュール，操作対象となる SUI Service モジュール，SUI Service の通信機能を担う SUI Service Container モジュール，SUI Service モジュールの情報を管理する SUI Directory Service モジュールの動作概要と機能について述べた．

第 5 章

身振り認識部の実装

本章では，身振り認識機能を持つコントローラ部の実装について述べる．まず，実装環境を述べ，利用者の身振りを認識する機能を持つコントローラ部である身振り認識部の構成を述べる．ついで，身振り認識部の実装について述べ，最後にまわす身振り取得アルゴリズムを解説する．

5.1 実装環境

身振り認識の実装として，Java 言語を使用した．また，身振り認識部では，画像認識ライブラリとして ARToolKit [8] の Java 言語実装である jARToolKit [15] を使用した．表 5.1 に実装環境を示す．

身振り認識では，Visual Marker と USB カメラを利用し，利用者の手指の動きを取得する．また，ボタンスイッチにより，利用者のうつ，つまむ，はなす動作を取得した．図 5.1 と図 5.2 に，USB カメラとボタンスイッチを示す．

5.2 身振り認識部の構成

本節では，身振り認識部の構成として，ソフトウェアとハードウェアの構成について述べる．身振り認識部は，SUI Framework において，SUI Controller を構成するコントローラ部として実装される．ソフトウェア構成では，身振り認識部を構成するソフトウェアとその機能について述べる．ハードウェア構成では，身振り認識部で用いるハードウェアについて述べる．

表 5.1 実装環境

項目	環境
CPU	Pentium M 1.7GHz
Memory	2GB
OS	Windows XP SP2
JDK	JDK1.4.2_06
USB カメラ	SONY PCGA-UVC11A

図 5.1　USB カメラ（PCGA-UVC11A）

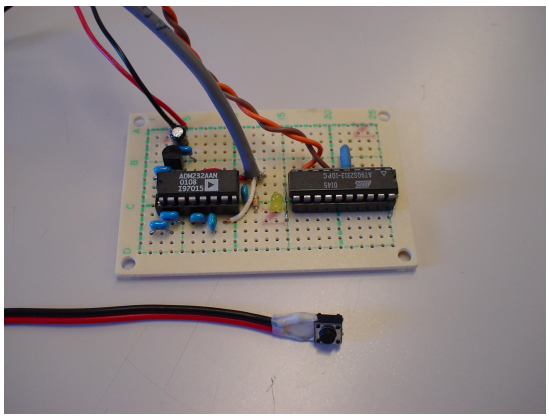


図 5.2　ボタン

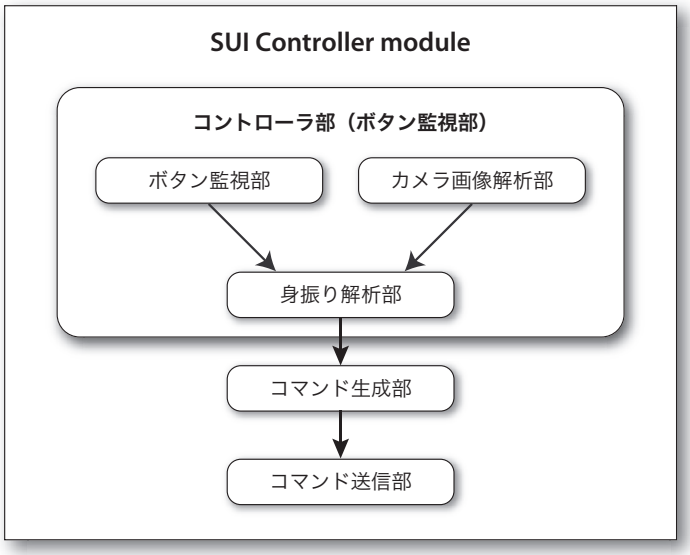


図 5.3　身振り認識部のソフトウェア構成図

5.2.1　ソフトウェア構成

本項では，身振り認識部のソフトウェア構成と身振り認識部を構成する各ソフトウェアの機能について述べる．身振り認識部のソフトウェアは，ボタン監視部，カメラ画像解析部，身振り解析部の 3 つから構成される．図 5.3 に身振り認識部のソフトウェア構成図を示す．以下で，それぞれについて解説する．

ボタン監視部

ボタン監視部はボタンスイッチの押下を監視する機能を持つ．ボタン監視部は，図 5.2 に示したボタンスイッチから，ボタンが押されたことを示す押下イベント，および押されたボタンが解放されたことを示す押下解放イベントを取得する．次に，取得した 2 つのイベントとその取得時間をもとに利用者のう

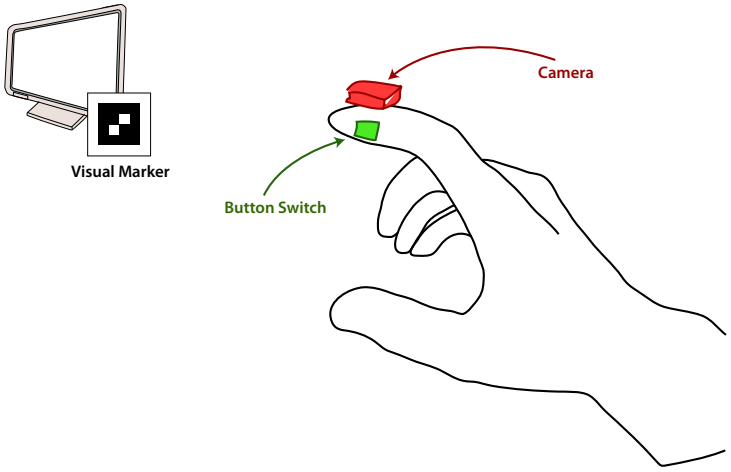


図 5.4　身振り認識部のハードウェア構成図

つ，つまむ，はなす動作を取得し，身振り解析部に伝える．ボタンスイッチとの通信はシリアル通信を介して行う．

カメラ画像解析部

カメラ画像解析部は，カメラ画像中の Visual Marker の位置と ID を取得する機能を持つ．カメラ画像解析部は図 5.1 に示した USB カメラから毎秒 30 フレームの頻度でカメラ画像を取得し，カメラ画像中の Visual Marker の位置と ID を検知する．次に，検知した Visual Marker の位置と ID を身振り解析部に渡す．Visual Marker の位置は画像中の座標として表現される．Visual Marker の ID は Visual Marker の模様とマッピングされ，模様毎に一意な ID となる．

身振り解析部

身振り解析部は，ボタン監視部とカメラ画像解析部から取得した情報をもとに，利用者の身振りを解析する機能を持つ．ボタン監視部からのボタン押下情報からはうつ，つまむ，はなすという 3 つの身振りを取得する．カメラ画像解析部からの Visual Marker 情報からはさす，まわすという 3 つの身振りを解析する．身振り解析部は解析した身振りをもとに，コマンド生成部で操作コマンドを生成する．

5.2.2　ハードウェア構成

身振り認識部のハードウェア構成は，図 5.4 に示すように，手指にカメラとスイッチボタンを取り付け，操作対象には Visual Marker を貼り付ける．カメラは USB 経由で PC と接続され，スイッチボタンはシリアル経由で PC と接続される． 身振り認識部は，手指に取り付けたカメラで Visual Marker を検知し，カメラ画像内の Visual Marker の動きを取得する．これにより，利用者のさす動作とまわす動作を取得する．また，手指に取り付けたスイッチボタンで利用者のうつ動作とつまむ，はなす動作を取得する．

5.3　身振り認識部の実装

本節では，身振り認識部の実装について述べる．身振り認識部はボタン監視部，カメラ画像解析部，身振り解析部で構成される．以下で，それぞれについて解説する．

表 5.2　ボタン押下コマンド

イベント	文字列
ボタンが押下された	c
押下状態にあるボタンが解放された	o

5.3.1　ボタン監視部

ボタン監視部は，シリアル通信経由で図 5.2 に示したスイッチボタンからイベントを取得する．表 5.2 にスイッチボタンから取得するイベントを示す．ボタン監視部は COM ポートを監視し，これら 2 つの文字列を待ち受ける．これら 2 つのイベントを受けて，ボタン監視部は身振り解析部の holdButton メソッド，ReleaseButton メソッド，hitButton メソッドを呼び出す．ボタン監視部のソースコードを付録．A に記載する

ボタン監視部では利用者のうつ，つまむ，はなすという 3 つの動作を検知する．ボタン監視部はボタンスイッチから取得する 2 つのイベントに加え，イベントの発生時間を用いることで，これら 3 つの動作を検知する．いかに，それぞれの動作の検知方法ついて，解説する．

うつ

うつ動作は指と人差し指の指先どうしを軽く接触させ，すぐに離す動作である．このため，スイッチボタンの押下イベントが発生してから，押下解放イベントが発生するまでの時間が 250 ミリ秒以下だった場合，うつ動作とする．

つまむ

つまみ動作は人差し指の指先どうしを軽く接触させ，その状態ほ保持する動作である．このため，スイッチボタンの押下イベントが発生し，250 ミリ秒経過した場合，つまみ動作とする．

はなす

はなす動作は人差し指の指先どうしを軽く接触させた状態から，指を離す動作である．このため，スイッチボタンの押下イベントが発生し，250 ミリ秒以上経過後，押下解放イベントが発生した場合にはなす動作とする．

5.3.2　カメラ画像解析部

カメラ画像解析部では，Visual Marker の検知に画像解析ライブラリとして，ARToolKit の Java 実装である jARToolKit を利用した．また，画像を取得するためのカメラには，Sony の PCGA-UVC11A [18] を用いた．カメラ画像解析部は jARToolKit を利用してカメラ画像中の Visual Marker を検知し，Visual Marker の 3 次元座標における変換行列を取得する．次に，取得した 3 次元座標における変換行列を 2 次元座標に変換し，640 × 480 ピクセルの画像に合わせてスケールさせる．このようにして取得した Visual Marker の 2 次元座標をカメラ画像解析部は身振り解析部に渡す．カメラ画像解析部のソースコードを付録．B に記載する．

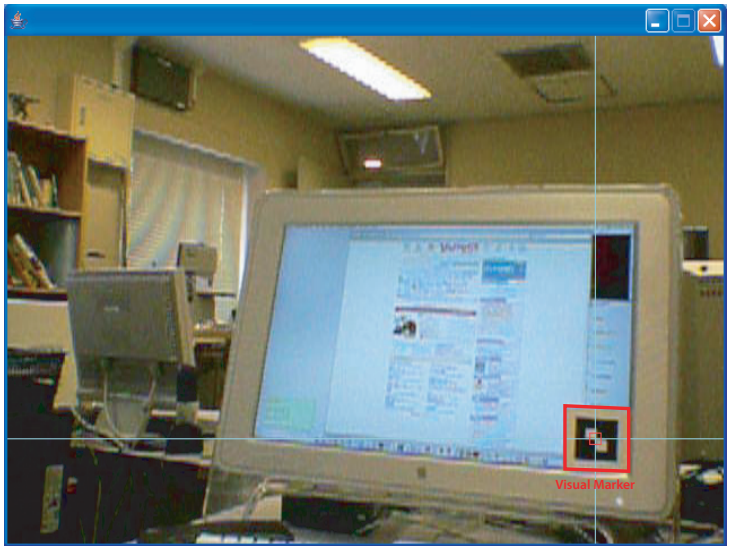


図 5.5　カメラ画像解析部のスクリーンショット

5.3.3　身振り解析部

身振り解析部では，ボタン監視部とカメラ画像解析部から取得した情報をもとに，利用者の身振りを取得する．身振り解析部はボタン監視部からはうつ，つまむ，はなす動作の情報を取得し，カメラ画像解析部からは Visual Marker の ID と 2 次元座標を取得する．身振り解析部はカメラ画像解析部からの情報を解析し，さす動作とまわす動作を検知する．以下で，さす動作とまわす動作の検知方法について解説する．

さす

さす動作は操作対象に向けて指さす動作である．このため，指に取り付けたカメラで Visual Marker を写し，Visual Marker の ID を取得することで操作対象を検知した場合にさす動作とする．カメラ画像内に n 個の Visual Marker が存在する場合，身振り解析部は身振り解析部に n 個の ID と座標をカメラ画像解析部から取得する．複数の Visual Marker を検知した場合は，よりカメラ画像の中心に近い Visual Marker をさす動作の対象とする．

まわす

まわす動作はつまんだ指先で円を描くようにまわす動作である．このため，身振り解析部はボタン監視部から取得したつまむ，はなす動作とカメラ画像解析部から取得した Visual Marker の 2 次元座標をもとに，利用者の指の動きを解析する．Visual Marker の 2 次元座標からまわす動作を取得するアルゴリズムについては，第 5.5 節で解説する．

5.4　まわす動作取得アルゴリズム

本節では，身振り解析部で実装した，まわす動作を取得するためのアルゴリズムについて述べる．まわす動作取得アルゴリズムでは以下の手順で回す動作を取得する．

1. 始点（つままれた瞬間の Visual Marker の座標）を取得する
2. 移動点（つままれた瞬間より後に取得した Visual Marker の座標）の通過エリアを算出する
3. 移動点と 4 直線の交点を算出する
4. 始点と交点をもとに中心点を算出する
5. 移動元の座標と異動先の座標の中心角を算出する
6. 中心角の大きさにより，まわす動作を取得する

以下で，それぞれの手順について解説する．

始点の取得

まず，身振り解析部はボタン監視部からつまみ動作を取得する．この時点で，カメラ画像解析部から Visual Marker の座標を取得していた場合，最も中心に近い Visual Marker を操作対象とみなし，その座標値を始点として設定する．

移動点の通過エリアを算出

次に，身振り解析部は始点から移動した Visual Marker の座標（以下，移動点）の通過エリアを算出する．通過エリアとは，図 5.6 に示すとおり，始点を中心とした 4 本の直線に区切られた 8 つのエリアである．身振り解析部は，移動点が通ってきたエリアと移動点が 8 直線と交わった点（以下，交点）をもとに，中心点を算出する．図 5.7，図 5.8，図 5.9 は移動点の移動軌跡パターンを表している．つまり，移動点の移動軌跡により，中心点の算出方法は 16 通り存在する．

移動点と 4 直線の交点を算出

身振り解析部は，移動点が 4 直線と交わるまで，移動点の通過エリアを算出し続ける．移動点と 4 直線が交わった場合，次の手順である中心点の算出にすすむ．

中心点の算出

身振り解析部は始点と交点をもとに，中心点を算出する．図 5.10 に示した例の場合，移動点はエリア 7 を通り，交点 Q に交わったため，中心角 Q の x 座標 m と y 座標 n は以下の式で算出する．

$$m = k$$

$$n = y$$

中心角の算出

身振り解析部は，図 5.11 に示したように，移動前の移動点 a と移動後の移動点 b を保持する．これにより，この 2 点と中心点を中心とした中心角を算出する．カメラ画像解析部からは，毎秒 30 回の間隔で Visual Marker の座標を取得するため，中心角の算出は毎秒 30 回行われる．

まわす動作の取得

身振り解析部は，算出した中心角が 10 度以上ならば，まわす動作として認識する．中心角の算出は毎秒 30 回行われるため，回す動作の取得も毎秒 30 回行われる．

5.5 まとめ

本章では，身振り認識機能を持つコントローラ部の実装について述べた．まず，実装環境を述べ，次に利用者の身振りを認識する機能を持つコントローラ部である身振り認識部のソフトウェア構成とハードウェア構成

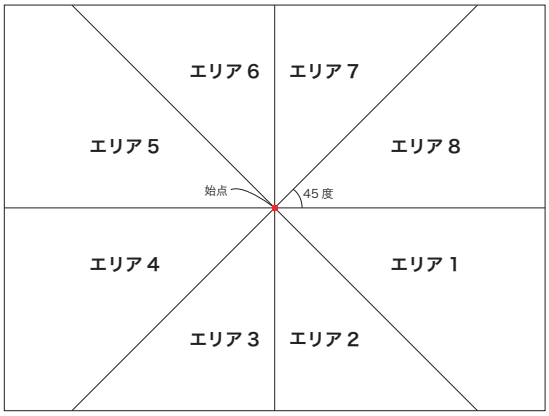


図 5.6　まわす動作取得アルゴリズム：エリア図

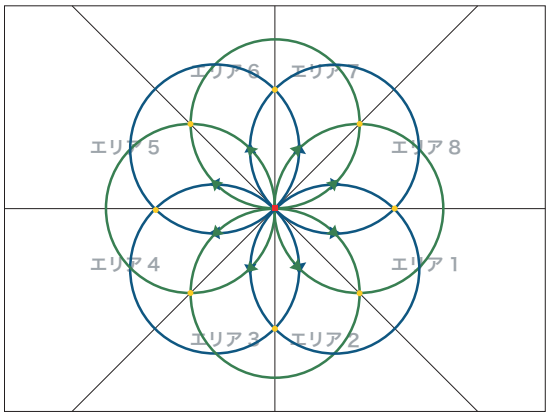


図 5.7　まわす動作取得アルゴリズム：
移動点の移動軌道全パターン

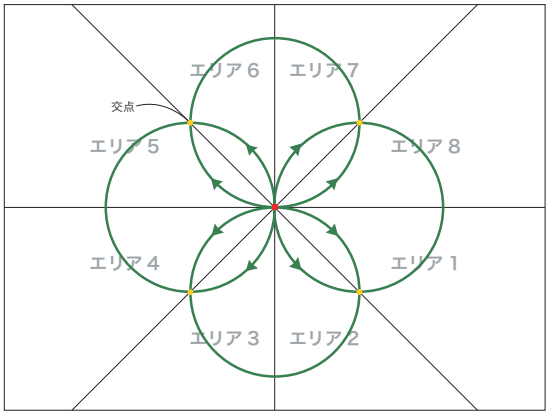


図 5.8　まわす動作取得アルゴリズム：
移動点の移動軌道パターン 1

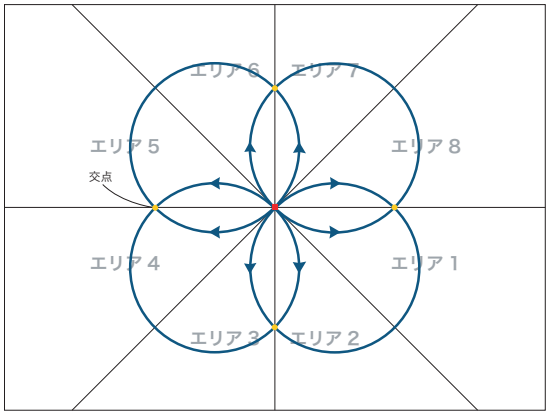


図 5.9　まわす動作取得アルゴリズム：
移動点の移動軌道パターン 2

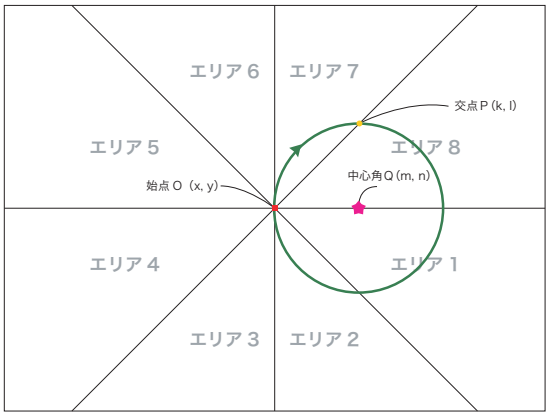


図 5.10　まわす動作取得アルゴリズム：
中心点の算出例

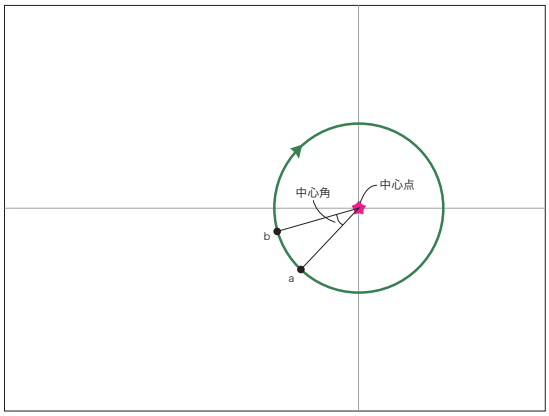


図 5.11　まわす動作取得アルゴリズム：
中心角の算出例

について述べた．身振り認識部はボタン監視部，カメラ画像解析部，身振り解析部の 3 つのソフトウェアで構成される．ハードウェアは，ボタンスイッチと USB カメラ，Visual Marker を利用した．ついで，身振り認識部の実装および，身振りの取得方法について述べ，最後にまわす身振り取得アルゴリズムを解説した．

第 6 章

本研究の評価

本章では，SUI Framework における身振りインタフェースを評価する．SUI Service として画像閲覧アプリケーションと写真立てアプリケーションを実装した．これらを操作対象として，本論文で設計，実装した統一的操作系による身振りインタフェースの学習コストを複数人の被験者による実験結果をもとに評価する．まず，実験環境と実験手順を述べ，ついで実験結果を述べた後，結果について考察する．

6.1 実験環境

本節では，本実験の実験環境について述べる．まず，操作対象となる画像閲覧アプリケーションと写真立てアプリケーションの機能を解説し，次に本実験の被験者について述べる．

6.1.1 本実験の操作対象

本実験では，操作対象として画像閲覧アプリケーションと写真立てアプリケーションを実装した．これらのアプリケーションの実装には SUI Framework を用いた．以下で，それぞれの機能について解説する．

画像閲覧アプリケーション

画像閲覧アプリケーションは，メモリースティックなどのフラッシュメモリメディアに保存された画像を閲覧するためのアプリケーションである．画像閲覧アプリケーションは，第 3.5.1 項で述べたコマンドのうち，Nomination コマンドと Change コマンド，Next/Previous コマンド，Start/End コマンドの 4 つに対応する．利用者に操作される内部変数は，画像閲覧アプリケーションが保持する画像一覧である．図 6.1 に示すとおり，画像は楕円状に並べられ，まわす動作によって表示位置を変更する．図 6.2 に操作されている画像閲覧アプリケーションを示す．また，画像閲覧アプリケーションは他の操作対象に対して，画像を送信する機能を持つ．利用者はまわす動作で送信したい画像を選択した後，うつ動作をし，送信元から送信先へつまんではなす動作をすることで，送信機能を利用できる．画像閲覧アプリケーションはさす動作を受けて，画面枠を赤く発光させるとともに Active!という文字列を表示し，利用者へフィードバックする．本実験では，画像を 10 枚用意し，画像閲覧アプリケーションで閲覧する．

写真立てアプリケーション

写真立てアプリケーションは，画像を表示するためのアプリケーションである．図 6.3 に写真立てアプリケーションを示す．写真立てアプリケーションは Source コマンドのみに対応する．本実験では，画像閲覧アプリケーションで選択した画像を送信し，表示するために写真立てアプリケーションを用い

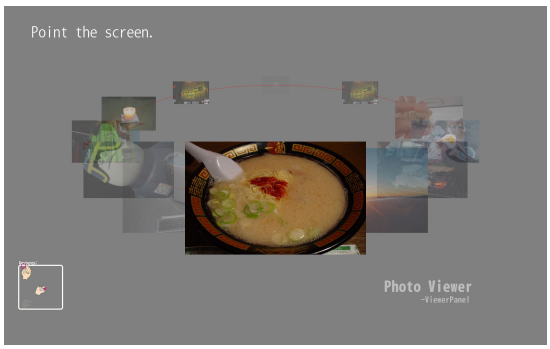


図 6.1　画像閲覧アプリケーションのスクリーンショット

図 6.2　まわされている状態の画像閲覧アプリケーションのスクリーンショット

図 6.3　写真立てアプリケーション

図 6.4　指された状態の写真立てアプリケーション

表 6.1　本実験の被験者分類と被験者数

被験者の分類	被験者数
身振りインタフェース熟練者	1
身振りインタフェース初心者	2

る．写真立てアプリケーションは，さす動作を受け，図 6.4 に示すとおり赤く発光することにより，利用者へフィードバックする．

6.1.2　本実験の被験者

本実験の被験者について表 6.1 を参照し，説明する．被験者は，身振りインタフェースに関する事前知識のない被験者 2 名と，身振りインタフェースの熟練ユーザ 1 名である．事前知識のない被験者は，コンピュータを日常的に使っているコンピュータ中級者を 2 名である．

6.2 実験手順

本実験は，身振りインタフェースについて簡単に説明した後，実験をするという手順で行った．身振りインタフェースについての説明とは，5 つの身振りとその身振りが持つ意味，操作対象が持つ機能についてである．図 6.5 に説明に用いたポスターを示す．学習コストの評価は，身振りインタフェースの初心者である被験者が決められた作業を終えるまでの所要時間を計測し，熟練者が同様の作業を終えるまでの所要時間と比較して評価する．

本実験において設定した作業は以下の通りである．被験者は以下の作業を 60 回繰り返し行い，一連の作業が終了するまでの所要時間を一回毎に計測する．

1. 画像閲覧アプリケーションを操作し，画像を選択する
2. 画像閲覧アプリケーションから写真立てアプリケーションへ画像を転送する

被験者はこれらの作業を身振りインタフェースを用いて作業する．作業 1 では，利用者はさす，つまむ，まわす，はなすという 4 つの身振りを用いて画像閲覧アプリケーションを操作する．作業 2 では，利用者はさす，うつ，つまむ，はなす，という 4 つの身振りを用いて，画像閲覧アプリケーションから写真立てアプリケーションへ画像を転送する．

6.3 実験結果

前述の 2 つの作業を終えるまでの所要時間と作業回数について，計測結果を図 6.6 と図 6.7 に示す．図 1 は身振りインタフェース初心者の所要時間全体に対して近似線をひいた図である．図 2 は身振りインタフェース初心者の所要時間のうち，ベストケースに対して近似線をひいた図である．所要時間の定義は，作業開始から写真立てアプリケーションに画像が表示されるまでに経過した時間とする．以上の実験結果をもとに身振りインタフェースの学習コストを評価，考察する．

本実験の結果から，本論文で設計，実装した統一的操作系にもとづく身振りインタフェース（以下，身振りインタフェース）の学習コストを評価する．身振りインタフェースの利用における学習内容は大きく 3 つに分けられる．1 つめがさす，うつ，つまむ，はなす，まわすという 5 種類のインタラクションの学習で．2 つめが前述の 5 種類のインタラクションと操作対象の機能とのマッピングを学習することである．3 つめが本論文で実装した身振り解析部の癖を学習することである．身振り解析部の癖とは，手指をまわす速度とまわす半径，操作対象との距離について，認識されやすい値と認識されにくい値があるということである．以下に，3 つの学習内容を整理する．

1. 5 種類の身振りの学習
2. 身振りと操作対象が持つ機能の学習
3. 身振り解析部が持つ癖の学習

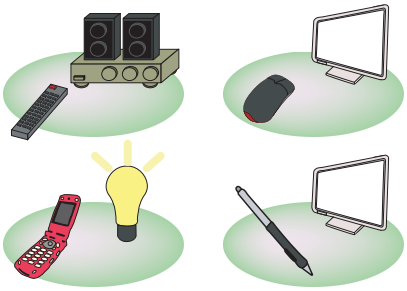
SUI:
仮想世界の情報を物理的に扱う実世界的インタフェース

http://www.ht.sfc.keio.ac.jp/kmsf/

Overview

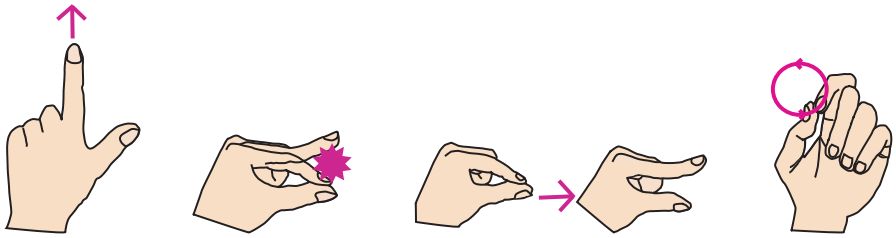
次世代ホームネットワークでは、さまざまな機器、情報、センサ群が連携したアプリケーションを動作させることができます。このような環境においては、アプリケーション及び、その機能が増加するにつれ、それぞれの機能に対応した操作方法も増加してゆきます。利用者がこれらの操作方法をすべて習得することは困難であり、統一的な操作系にもとづく操作インタフェースが求められています。また、次世代ホームネットワークでは操作対象は実空間に存在するため、実世界指向インタフェースで操作できることが望まれます。

本研究では操作方法として手指によるジェスチャを用いることで、操作対象への直接操作を実現します。また、操作方法に用いるジェスチャをさす、うつ、つまむ・はなす まわすの4つに限定することで 利用者が習得しやすく 統一的な操作系にもとづいた操作インタフェースを実現します。

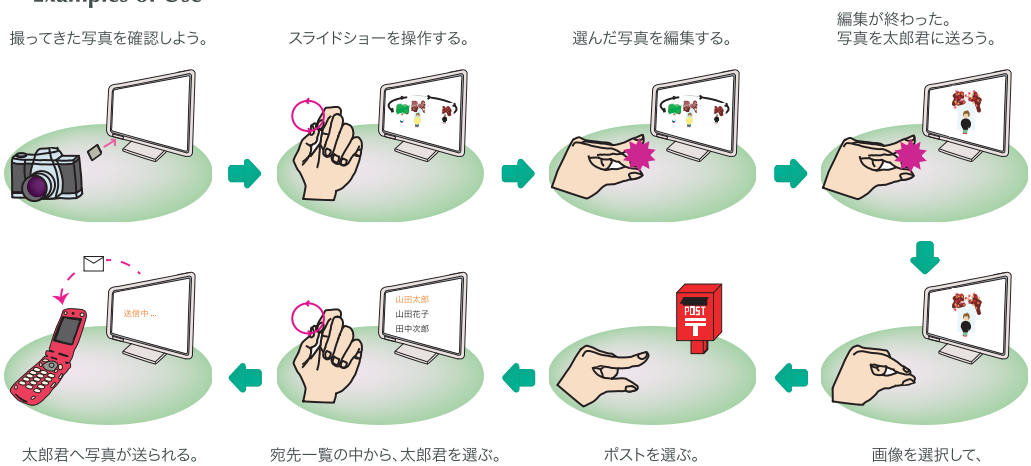


アプリケーションごとに操作方法が異なる

Explanation of Gesture



Examples of Use



関係組織　日立
※このプロジェクトは、文部科学省科学技術振興調整費「先導的研究等の推進」プログラム「横断的科学によるユビキタス情報社会の研究」および、
文部科学省 21 世紀 COE プロジェクト　学際・複合・新領域「次世代メディア・知的社会基盤」の下に行われています。

図 6.5　説明に用いたポスター

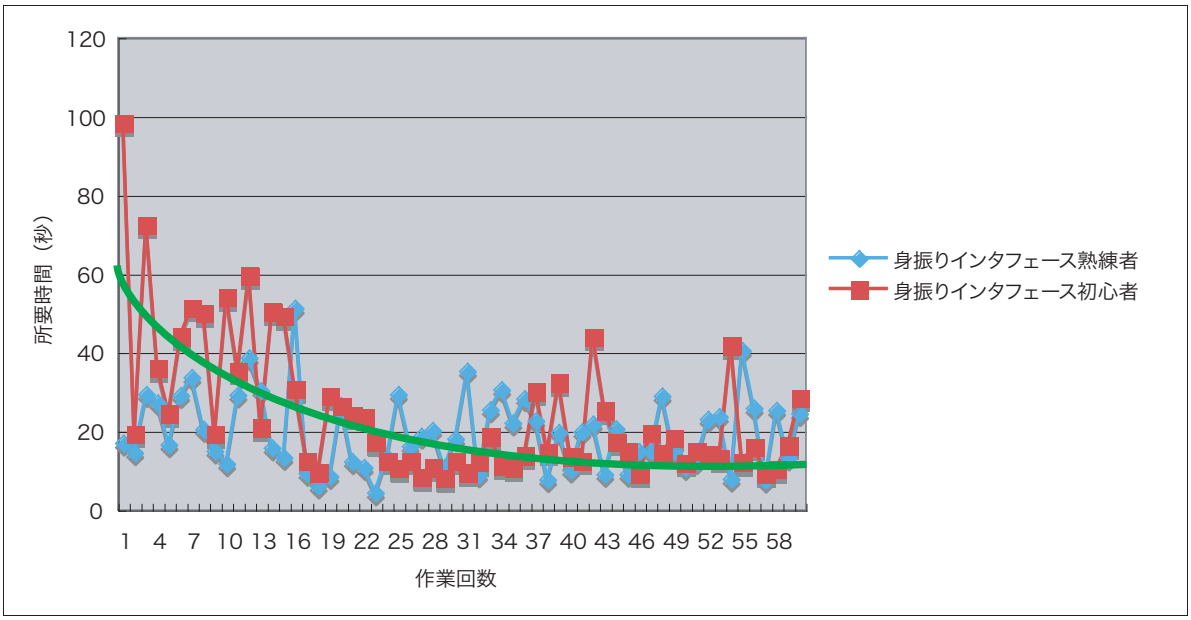


図 6.6　計測結果:全体

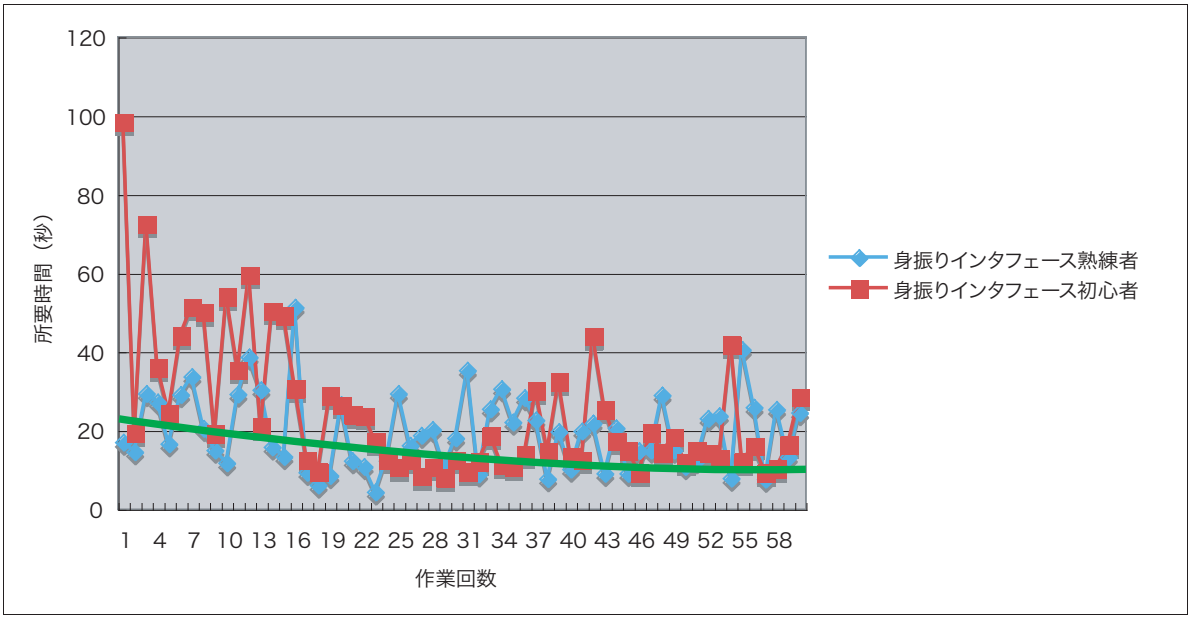


図 6.7　計測結果:ベストケース

6.4 考察

本節では，第 6.3 節で述べた 3 つの学習内容をふまえて，図 6.6 と図 6.7 に示す測定結果から，身振りインタフェースの学習コストについて考察する．

図 6.6 のグラフでは，初心者の作業所要時間が作業回数 20 回を境に 10 秒から 15 秒程度に収束をはじめることを読み取れる．これは熟練者の作業所要時間と比べても十分短いといえる．このことから，身振りインタフェースの初心者は，20 回程度の使用で前述の 3 つの学習を終えると考えられる．

次に，図 6.6 のグラフでは，作業回数 20 回まで，初心者の作業所要時間の回ごとの差がに非常に大きいことが読み取れる．これは，前述の身振り解析部の持つ癖が原因である．評価実験中において，身振りの認識がうまくいかず，初心者が作業をやり直す光景が見受けられた．初心者の作業所要時間のうち，作業回数 20 回を超えていなくても，所要時間の収束値に近い所要時間で作業を終えている回があるのに対し，その次の回で，収束値の倍以上の時間で作業を終えている．これは身振りの認識エラーにより，初心者が作業をやり直すことが原因である．このことから，作業回数 20 回未満における作業所要時間のばらつきは，身振り解析部の癖を学習するコストが原因であり，身振り解析部の実装を改善することにより解決できるといえる．身振り解析部の実装の問題点と解決策については，第 7.1.1 項で述べる．

また，初心者の作業所要時間のうち，ベストケースに対して近似線をひいた図 6.7 からは，作業回数 2 回目に収束値に近い 20 秒という値が読み取れる．これに加え，作業所要時間全体が収束を始める 20 回を超えるまでに，収束値に近い値が 6 回計測されている．このことに加え，前述の身振り解析部の癖を学習するコストの存在から，第 6.3 項で述べた 3 つの学習のうち，1 つめの 5 種類の身振りの学習と 2 つめの身振りと操作対象が持つ機能の学習は 1 回の作業で終えていると言える．このことから，本研究で提案した統一的操作系による身振りインタフェースは，1 回の使用で新しい操作対象の操作方法を学習できると言える．

6.5 まとめ

本章では，SUI Framework における身振りインタフェースを評価した．SUI Service として画像閲覧アプリケーションと写真立てアプリケーションを実装し，これらを操作対象として実験環境を設定した．この実験環境のもとで，本論文で設計，実装した統一的操作系による身振りインタフェースの学習コストを複数人の被験者による実験結果をもとに評価した．この結果，身振りインタフェースの実装改善の必要性と，身振りインタフェースの操作方法は 1 回の使用で学習できることを示した．

第 7 章

結論

7.1 今後の課題

本研究は主に以下の 2 つについて，今後の課題として考慮しなければならない．

7.1.1 身振り解析部の実装改善

身振り解析部の実装を改善する必要がある．第 6.4 節で指摘したとおり，現在の身振り解析部の実装では，認識率が十分でない．身振り解析部の現在の実装における問題点と解決策をいかに整理する．

Visual Marker の認識率

Visual Marker の認識率を改善する必要がある．現在の jARToolKit を用いた実装では，1 辺 6cm の Visual Marker の場合，最大 150cm までが実用上十分な認識率となる．150cm より離れた場所から操作対象を操作するために，Visual Marker の認識率を改善する必要がある．

カメラの画角

Visual Marker 検知のために用いるカメラの画角を広げる必要がある．Visual Marker の認識率の問題により，操作対象からは最大 150cm しか離れることができない．このため，本実装で利用した PCGA-UVC11A の画角では，まわす動作における手指が描く円の直径は最大 7cm 程度に制約される．手指が描く円の直径が，7cm を超えると，身振り解析部はまわす動作を解析できない．これを解決するために，カメラの画角を広げる必要がある．

カメラのシャッタースピード

カメラのシャッタースピードについて考慮しなければならない．本実装で利用した PCGA-UVC11A は，光量が少ない場所ではシャッタースピードが遅くなるため，カメラ画像がぶれ，Visual Marker が認識できない．光量が少ない場所で身振りインタフェースが利用するために，カメラのシャッタースピードについて考慮する必要がある．

中心点算出方法

まわす動作取得のための中心点算出方法を改善する必要がある．現在の実装では，利用者が指をまわしはじめてから，90 度動かさないと，まわす身振りの認識が開始されない．このため，利用者の入力に対して，即時的に操作対象が動作できない．この問題点を解決するために，中心点算出方法を改善する必要がある．

7.1.2 フィードバックを考慮したフレームワーク

利用者へのフィードバックについて，考慮する必要がある．第 6.4 節で指摘したとおり，現在の実装では，利用者の身振りを十分に認識できないという問題がある．第 7.1.1 項で実装に関する解決策を述べた．しかし，実装の改善とは別に，認識エラーを利用者にフィードバックする機能があると，利用者はエラーを発生を抑え，させずに作業することができる．このため，SUI Framework の改善点として，フィードバックの考慮を今後の課題とする．

7.2 本論文のまとめ

本論文では，ホームコンピューティング環境において，様々な操作対象の操作に際し，統一的な操作系を用いて身振り入力を行う身振り入力インタフェースを提案した．統一的な操作系による身振り入力インタフェースにより，ホームコンピューティング環境の操作対象の数の増加，操作対象の種類の増加，操作対象の機能の変化からくる利用者の学習コスト増加問題を解決できる．

本論文では，統一的な操作系による身振り入力インタフェースを実現するために，統一的な操作系を持つ入力インタフェースフレームワークとして，SUI Framework を設計し，そのソフトウェアの構成と機能，実装について述べた．また，利用者の身振りを認識し，操作対象への入力へ利用するために，SUI Framework のモジュールとして，身振り解析部を実装した．

本研究の評価として，統一的な操作系による身振り入力インタフェースの被験者による評価実験を行った．統一的な操作系による身振り入力インタフェースを用いて，設定した作業を終えるまでの所要時間を計測することで，身振り入力インタフェースの学習コストについて評価した．その結果，身振りインタフェースの実装改善の必要性と，身振りインタフェースの操作方法は 1 回の使用で学習できることを示した．

謝辞

本研究の機会を与えてくださり，絶えず丁寧なご指導を賜りました，慶應義塾大学環境情報学部教授徳田英幸教授に深く感謝致します．また，貴重なご助言を頂きました應義塾大学政策・メディア研究科助教授高汐一紀博士に深く感謝致します．

また，慶応義塾大学徳田研究室の諸先輩方には，折りに触れ貴重な示唆や御指導を頂きました．特に，政策・メディア研究科高橋元氏，門田昌也氏には，本研究の草稿時から多くの助言と励ましを頂きました．政策・メディア研究科講師岩井将行博士，同研究科後期博士課程由良淳一氏には，本論文執筆にあたって多くの励ましとご指導を得ました．ここに，深い感謝の意を表します．

最後に，研究生活を支えてくれた家族，研究の日々を共に過ごした KMSF 研究グループ，その他の友人に深く感謝し，謝辞と致します．

山崎 俊作

参考文献

[1] R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, Vol. 6, pp. 355–385, aug 1997.

[2] R. A. Bolt. 'put-that-there': Voice and gesture at the graphics interface. In *Proceedings of Siggraph 80*, jul 1980.

[3] World Wide Web Consortium. Extensible markup language(xml) 1.0, feb 1999.

[4] Object Management Group. Unified modeling language. http://www.uml.org/.

[5] Sun Microsystems Inc. Jini network technology. http://www.sun.com/software/jini/.

[6] H. Ishii. Tangible bits: User interface design towards seamless integration of digital and physical worlds (in japanese). In *IPSJ Magazine*, pp. 222–229, 2002.

[7] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of CHI '97*, pp. 234–241, 1997.

[8] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *2nd International Workshop on Augmented Reality (IWAR 99)*, 1999.

[9] J. Nakazawa, Y. Tobe, and H. Tokuda. On dynamic service integration in vna architecture. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 7, No. E84-A, pp. 1610–1623, jul 2001. http://www.ht.sfc.keio.ac.jp/~jin/VNA/.

[10] D. A. Norman. *The Psychology of Everyday Things*. Basic Books, 1988.

[11] Universal Plug and Play Forum. Universal plug and play (upnp). http://www.upnp.org/.

[12] J. Rekimoto. Gesturewrist and gesturepad: Unobtrusive wearable interaction devices. In *IEEE International Symposium on Wearable Computer*, 2001.

[13] K. Ryokai, S. Marti, and H. Ishii. I/o brush: Drawing with everyday objects as ink. In *Proceedings of CHI 2004*, apr 2004.

[14] Sony, Matsushita, Philips, Thomson, Hitachi, Toshiba, Sharp, and Grundig. Specification of the home audio/video interoperability (havi) architecture, may 1998. http://www.havi.org/home.html.

[15] the jARToolKit Team. jartoolkit - a java-binding to the artoolkit. http://www.c-lab.de/jartoolkit/.

[16] K. Tsukada and M. Yasumura. Ubi-finger: Gesture input device for mobile use. In *APCHI*, Vol. 1, pp. 388–400, 2002.

[17] M. Weiser. The computer for the 21st century. *Scientific American*, Vol. 265, No. 3, pp. 66–75, jan 1991.

[18] ソニー株式会社. ビジュアルコミュニケーションカメラ pcga-uvc11a.

http://www.ecat.sony.co.jp/computer/vaio/acc/index.cfm?PD=17110&KM=PCGA-UVC11A.

[19] 福本雅朗, 外村佳伸. “ 指釦 ”: 手首装着型コマンド入力機構. 情報処理学会論文誌, Vol. 40, No. 2, feb 1999.

[20] 株式会社スギヤマエレクトロン. クロッサム 2+usb. http://www.sugi-ele.co.jp/index.htm.

[21] 海保博之, 原田悦子, 黒須正明. 認知的インタフェース. 新曜社, 1991.

[22] 宇野裕史, 花田恵太郎, 豆田憲治, 川尻百恵, 樫原潤三, 吉川達夫, 中川浩和, 神井美和, 笠原洋子. ユニバーサルコントローラの開発. Technical report, シャープ株式会社, dec 1999.

[23] 岩崎陽平, 河口信夫, 稲垣康善. Touch-and-connect：ユビキタス環境における接続指示フレームワーク. 情報処理学会論文誌, Vol. 45, No. 12, pp. 2642–2654, dec 2004.

[24] 黒川隆夫. ノンバーバルインタフェース. オーム社, 1994.

付録 A

ボタン監視部のソースコード

```
/*
 * ボタン監視部
 *
 */
package jp.kmsf.sui.controller.gesture;

import java.io.IOException;
import java.io.InputStream;
import java.util.Timer;
import java.util.TooManyListenersException;

import javax.comm.CommPortIdentifier;
import javax.comm.NoSuchPortException;
import javax.comm.PortInUseException;
import javax.comm.SerialPort;
import javax.comm.SerialPortEvent;
import javax.comm.SerialPortEventListener;
import javax.comm.UnsupportedCommOperationException;

/**
 * @author Shunsaku Yamazaki(sunsaku@ht.sfc.keio.ac.jp>
 */

public class PushButtonObserver extends Thread implements SerialPortEventListener {

    private CommPortIdentifier commPortIdentifier;
    private SerialPort serialPort;
    private InputStream inputStream;
    private GestureHandler gestureHandler;
    private boolean isAlive;
    private long time;
    private Timer timer;

    public PushButtonObserver(String com, GestureHandler gestureHandler)
    throws ComNotOpenedException {
        this.gestureHandler = gestureHandler;
        this.isAlive = true;
        this.time = 0;
        try {
            this.commPortIdentifier = CommPortIdentifier.getPortIdentifier(com);
            this.serialPort = (SerialPort)this.commPortIdentifier.open(PushButton", 2000);
            this.inputStream = this.serialPort.getInputStream();
            this.serialPort.addEventListener(this);
            this.serialPort.notifyOnDataAvailable(true);
            this.serialPort.setSerialPortParams(
                    19200,
                    SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1,
                    SerialPort.PARITY_NONE);
        }
        catch (NoSuchPortException e) {
            throw new ComNotOpenedException();
        }
        catch (PortInUseException e) {
            throw new ComNotOpenedException();
        }
        catch (IOException e) {
            throw new ComNotOpenedException();
```

```
        }
        catch (TooManyListenersException e) {
            throw new ComNotOpenedException();
        }
        catch (UnsupportedCommOperationException e) {
            throw new ComNotOpenedException();
        }
    }

    public void serialEvent(SerialPortEvent serialPortEvent) {
        switch (serialPortEvent.getEventType()) {
            case SerialPortEvent.BI:
            case SerialPortEvent.OE:
            case SerialPortEvent.FE:
            case SerialPortEvent.PE:
            case SerialPortEvent.CD:
            case SerialPortEvent.CTS:
            case SerialPortEvent.DSR:
            case SerialPortEvent.RI:
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                break;
            case SerialPortEvent.DATA_AVAILABLE:
                byte[] readBuffer = new byte[20];
                try {
                    int numBytes = 0;
                    while (0 < this.inputStream.available()) {
                        numBytes =+ inputStream.read(readBuffer);
                    }
                    String c = new String(readBuffer, 0, numBytes);
                    // スイッチが押された
                    if (c.equals("o")) {
                        this.time = System.currentTimeMillis();
                        this.timer = new Timer();
                        this.timer.schedule(
                            new PushButtonTimer(this.gestureHandler),
                            250);
                    }
                    // スイッチが離された
                    else if (c.equals("c")) {
                        // はなされた
                        if (250 < System.currentTimeMillis() - this.time) {
                            this.gestureHandler.releaseButton();
                        }
                        // うたれた
                        else {
                            this.timer.cancel();
                            this.gestureHandler.hitButton();
                        }
                    }
                }
                catch (IOException e) {}
                break;
        }
    }

    public void run() {
        while (true) {
            if (!this.isAlive) {
                break;
            }
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {}
        }
    }

    public void close() {
        this.serialPort.close();
        this.isAlive = false;
    }
}

/*
 * ボタン監視部タイマー
 *
 */
package jp.kmsf.sui.controller.gesture;
```

```
import java.util.TimerTask;

/**
 * @author Shunsaku Yamazaki(sunsaku@ht.sfc.keio.ac.jp)
 */
public class PushButtonTimer extends TimerTask {

    private GestureHandler gestureHandler;

    public PushButtonTimer(GestureHandler gestureHandler) {
        this.gestureHandler = gestureHandler;
    }
    public void run() {
        this.gestureHandler.holdButton();
    }
}
```

付録 B

カメラ画像解析部のソースコード

```
/*
 * Visual Marker 検知
 *
 */
package jp.kmsf.sui.controller.gesture;

import java.awt.Point;
import java.util.ArrayList;

import net.sourceforge.jartoolkit.core.JARToolKit;
import net.sourceforge.jartoolkit.videoinput.videocapturing.JARVideo;

/**
 * @author Shunsaku Yamazaki(sunsaku@ht.sfc.keio.ac.jp)
 */
public class MarkerHandler {

    private JARToolKit jarToolKit;
    private JARVideo jarVideo;
    private GestureHandler gestureHandler;
    private VideoFrame videoFrame;

    private int width;
    private int height;
    private int size;
    private int[] dataArray;

    private String camera_para = "C:\\jARToolKit_v2.0\\data\\camera_para.dat";
    private String pattern_calib = "C:\\jARToolKit_v2.0\\data\\pattern\\patt.calib";
    private String pattern_hiro = "C:\\jARToolKit_v2.0\\data\\pattern\\patt.hiro";
    private String pattern_kanji = "C:\\jARToolKit_v2.0\\data\\pattern\\patt.kanji";
    private String pattern_sample1 = "C:\\jARToolKit_v2.0\\data\\pattern\\patt.sample1";
    private String pattern_sample2 = "C:\\jARToolKit_v2.0\\data\\pattern\\patt.sample2";
    private int id_calib;
    private int id_hiro;
    private int id_kanji;
    private int id_sample1;
    private int id_sample2;

    public MarkerHandler() throws InstantiationException, ComNotOpenedException {
        // JARVideo の設定
        this.jarVideo = JARVideo.create("inputDevice=WDM_CAP,flipV,showDlg");
        this.jarVideo.setFlippedImage(true);
        this.jarVideo.grabFrame();
        this.width = this.jarVideo.getWidth();
        this.height = this.jarVideo.getHeight();
        this.size = (int)this.jarVideo.getBufferSize();
        this.dataArray = new int[(size>>2)];
        this.jarVideo.getBuffer(this.dataArray);

        // JARToolKit の設定
        this.jarToolKit = JARToolKit.create();
        this.jarToolKit.paramLoad(this.camera_para);
        this.jarToolKit.paramChangeSize(this.width, this.height);
        this.jarToolKit.initCparam();

        // Pattern の読み込み
        this.id_calib = this.jarToolKit.loadPattern(this.pattern_calib);
```

```
    this.id_hiro = this.jarToolKit.loadPattern(this.pattern_hiro);
    this.id_kanji = this.jarToolKit.loadPattern(this.pattern_kanji);
    this.id_sample1 = this.jarToolKit.loadPattern(this.pattern_sample1);
    this.id_sample2 = this.jarToolKit.loadPattern(this.pattern_sample2);

    // GestureHandler の設定
    this.gestureHandler = new GestureHandler(this.width, this.height, this);
    new PushButtonObserver("COM4", this.gestureHandler).start();

    // VideoFrame の設定
    this.videoFrame = new VideoFrame(this.width, this.height);

    System.out.println("## Gesture start ##");
    System.out.println("Width: " + this.width);
    System.out.println("Height: " + this.height);
    System.out.println("BitCount: " + this.jarVideo.getBitCount());
    System.out.println("BufferSize: " + this.size);
    System.out.println("## Gesture start ##");
}

public void grab() {
    this.jarVideo.getNextBuffer(this.dataArray);
    int[] idArray = this.jarToolKit.detectMarkerLite(this.dataArray, 100);

    // パターンの ID と位置を取得した
    if (0 < idArray.length) {
        ArrayList coordinatesList = new ArrayList();
        int count = 0;

        for (int i = 0; i < idArray.length; i++) {
            // 正しい ID を取得した
            if (-1 < idArray[i]) {
                double pattern_matrix[] = this.jarToolKit.getTransMatrix(idArray[i], 40, 0.0f, 0.0f);

                double x = (pattern_matrix[12] * 761.61959 + pattern_matrix[14] * 330.0) / pattern_matrix[14];
                double y = (pattern_matrix[13] * 844.88753 + pattern_matrix[14] * 196.0) / pattern_matrix[14];
                coordinatesList.add(new Point((int)x, (int)y));
                count++;
            }
        }
        int[] ids = new int[count];
        count = 0;
        for (int i = 0; i < idArray.length; i++) {
            if (-1 < idArray[i]) {
                ids[count] = idArray[i];
                count++;
            }
        }

        Point[] coordinates = (Point[])coordinatesList.toArray(new Point[coordinatesList.size()]);
        // gesture をハンドリング
        // 渡す情報としては、パターン ID と Point のマップ配列
        if (coordinates.length <= 0) {
            this.gestureHandler.handle(null, null);
            this.videoFrame.display(this.dataArray);
        }
        else {
            this.gestureHandler.handle(ids, coordinates);
            // video を表示
            this.videoFrame.display(this.dataArray, coordinates);
        }
    }
    // パターン発見できず
    else {
        this.gestureHandler.handle(null, null);
        this.videoFrame.display(this.dataArray);
    }
}

public String getPatternById(int id) throws NoSuchPatternException {
    if (id == this.id_calib) return "patt.calib";
    else if (id == this.id_hiro) return "patt.hiro";
    else if (id == this.id_kanji) return "patt.kanji";
    else if (id == this.id_sample1) return "patt.sample1";
    else if (id == this.id_sample2) return "patt.sample2";

    throw new NoSuchPatternException();
}
```

```
    public String getIdById(String id) throws NoSuchPatternException {
        if (id.equals("patt.hiro")) return "10002";
        if (id.equals("patt.kanji")) return "10003";
        if (id.equals("patt.sample1")) return "10004";
        if (id.equals("patt.sample2")) return "10005";
        if (id.equals("patt.calib")) return "10006";
        throw new NoSuchPatternException();
    }
}

/*
 * カメラ映像表示
 *
 */
package jp.kmsf.sui.controller.gesture;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.geom.Line2D;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;

import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

/**
 * @author Shunsaku Yamazaki(sunsaku@ht.sfc.keio.ac.jp)
 */
public class VideoFrame extends JFrame {

    private BufferedImage bufferedImage;
    private Graphics2D graphics2D;
    private WritableRaster writableRaster;
    private int width;
    private int height;

    public VideoFrame(int width, int height) {
        this.width = width;
        this.height = height;

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
            SwingUtilities.updateComponentTreeUI(this);
        }
        catch (ClassNotFoundException e) {}
        catch (IllegalAccessException e) {}
        catch (InstantiationException e) {}
        catch (UnsupportedLookAndFeelException e) {}
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(width, height);
        this.setResizable(false);
        this.setVisible(true);

        this.graphics2D = (Graphics2D)this.getGraphics();
        this.graphics2D.setBackground(Color.black);
        this.graphics2D.clearRect(0, 0, this.width, this.height);

        this.bufferedImage = new BufferedImage(this.width, this.height, BufferedImage.TYPE_INT_RGB);
        this.writableRaster = bufferedImage.getRaster();
    }

    // ビデオデータを表示する
    public void display(int dataArray[]) {
        this.writableRaster.setDataElements(0, 0, this.width, this.height, dataArray);
        this.graphics2D.drawImage(this.bufferedImage, 0, 0, this);
    }

    // ビデオデータと一つのポインタを表示する
    public void display(int[] dataArray, Point coordinate) {
        this.writableRaster.setDataElements(0, 0, this.width, this.height, dataArray);
        Graphics2D subGraphics2D = (Graphics2D)this.bufferedImage.getGraphics();
        subGraphics2D.setColor(new Color(153, 204, 204));
        subGraphics2D.draw(new Line2D.Double(coordinate.getX(), 0, coordinate.getX(), this.height));
        subGraphics2D.draw(new Line2D.Double(0, coordinate.getY(), this.width, coordinate.getY()));
        subGraphics2D.setColor(new Color(255, 51, 51));
```

```
            subGraphics2D.draw(new Rectangle2D.Double(coordinate.getX() - 5, coordinate.getY() - 5, 10, 10));
            this.graphics2D.drawImage(this.bufferedImage, 0, 0, this);
        }

        // ビデオデータと複数のポインタを表示する
        public void display(int[] dataArray, Point coordinate[]) {
            this.writableRaster.setDataElements(0, 0, this.width, this.height, dataArray);
            Graphics2D subGraphics2D = (Graphics2D)this.bufferedImage.getGraphics();
            for (int i = 0; i < coordinate.length; i++) {
                subGraphics2D.setColor(new Color(153, 204, 204));
                subGraphics2D.draw(new Line2D.Double(coordinate[i].getX(), 0, coordinate[i].getX(), this.height));
                subGraphics2D.draw(new Line2D.Double(0, coordinate[i].getY(), this.width, coordinate[i].getY()));
                subGraphics2D.setColor(new Color(255, 51, 51));
                subGraphics2D.draw(new Rectangle2D.Double(coordinate[i].getX() - 5, coordinate[i].getY() - 5, 10, 10));
            }
            this.graphics2D.drawImage(this.bufferedImage, 0, 0, this);
        }

        // ビデオデータと ID と複数のポインタを表示する
        public void display(int[] dataArray, int[] ids, Point coordinate[]) {
            this.writableRaster.setDataElements(0, 0, this.width, this.height, dataArray);
            Graphics2D subGraphics2D = (Graphics2D)this.bufferedImage.getGraphics();
            for (int i = 0; i < coordinate.length; i++) {
                subGraphics2D.setColor(new Color(153, 204, 204));
                subGraphics2D.draw(new Line2D.Double(coordinate[i].getX(), 0, coordinate[i].getX(), this.height));
                subGraphics2D.draw(new Line2D.Double(0, coordinate[i].getY(), this.width, coordinate[i].getY()));
                subGraphics2D.setColor(new Color(255, 51, 51));
                subGraphics2D.draw(new Rectangle2D.Double(coordinate[i].getX() - 5, coordinate[i].getY() - 5, 10, 10));
            }
            this.graphics2D.drawImage(this.bufferedImage, 0, 0, this);
        }
    }
```

付録 C

身振り解析部のソースコード

```
/*
 * 身振り解析部
 *
 */
package jp.kmsf.sui.controller.gesture;

import java.awt.Point;

/**
 * @author Shunsaku Yamazaki(sunsaku)
 */
public class GestureHandler {

    private GestureRecognizer gestureRecognizer;
    private GestureDispatcher2 gestureDispatcher;
    private MarkerHandler markerHandler;
    private int width;
    private int height;
    private int radius;

    private int presentId;    // 現在の Id
    private int startId;    // Start したときの Id

    private long nominationTime; // ささされた時間

    // 指されているかどうか
    // つままれているかどうか
    // 中心がでているかどうか
    private boolean isNomination;
    private boolean isHold;
    private boolean isStart;

    public GestureHandler(int width, int height, MarkerHandler markerHandler) {
        this.width = width;
        this.height = height;
        this.markerHandler = markerHandler;
        this.gestureRecognizer = new GestureRecognizer();
        this.gestureDispatcher = new GestureDispatcher2(this.markerHandler);
        this.isNomination = false;
        this.presentId = -1;
        this.startId = -1;
        this.isHold = false;
        this.isStart = false;
    }
    private int hoge;
    public void handle(int[] idArray, Point[] coordinates) {

        // pattern が発見されなかった
        if (idArray == null || coordinates == null) {
            if (hoge > 100) hoge = 0;
            if (150 < (System.currentTimeMillis() - this.nominationTime)) {
                this.isNomination = false;
                this.presentId = -1;
            }
        }
        // pattern が発見された
        else {
            // nomination 判定
```

```
        int number = this.isNomination(idArray, coordinates);
        // 見つかった
        if (0 <= number) {
            this.isNomination = true;

            // 前回と同じ ID である。
            if (this.presentId == idArray[number]) {
            }
            // 前回とは違う ID である
            // エラーである可能性が高い
            else {
                // 誤認識の可能性が高い場合
                if (150 < (System.currentTimeMillis() - this.nominationTime)) {
                }
                this.presentId = idArray[number];
            }


            // Nomination を送る
            this.gestureDispatcher.nomination(this.presentId);

            // もし、Hold していたら
            if (isHold) {
                if (!this.isStart) {
                    this.gestureRecognizer.setOrigin(coordinates[number]);
                    this.isStart = true;
                }
                try {
                    if (this.gestureRecognizer.analogize(coordinates[number]) == GestureRecognizer.NEXT) {
                        this.gestureDispatcher.next(this.presentId);
                    }
                    else {
                        this.gestureDispatcher.previous(this.presentId);
                    }
                } catch (GestureNotRecognizedException e) {}

                // Point を送る
                this.gestureDispatcher.point(this.presentId, coordinates[number]);
            }
        }
        // 中心から離れすぎている
        else {
            this.isNomination = false;
            this.presentId = -1;
        }
    }
}

// Nomination の判定
// 一番近い点を Nomination とする
private int isNomination(int[] idArray, Point[] coordinates) {
    if (idArray.length <= 0 || coordinates.length <= 0) {
        return -1;
    }

    int min = 0;
    double sum = Math.pow((coordinates[0].getX() - 320), 2) + Math.pow((coordinates[0].getY() - 240), 2);
    for (int i = 1; i < idArray.length; i++) {
        // より中心に近いほうを選ぶ
        double tmp = Math.pow((coordinates[i].getX() - 320), 2) + Math.pow((coordinates[i].getY() - 240), 2);
        if (tmp < sum) {
            min = i;
            sum = tmp;
        }
    }
    // リース時間を計る
    this.nominationTime = System.currentTimeMillis();
    return min;
}

public void holdButton() {
    // 既に Nomination していた場合
    if (this.isNomination) {
        this.gestureDispatcher.start(this.presentId);
        this.startId = this.presentId;
        this.isHold = true;
    }
}
```

```
    public void releaseButton() {
        // 既に Nomination していて、今 Nomination しているサービスが Start されていた場合
        if (this.isHold && this.isNomination && this.isStart && this.presentId == this.startId) {
            this.gestureDispatcher.end(this.presentId);
        }
        // 既に Nomination していて、Start されていたサービスが現在 Nomination しているサービスと違った場合
        else if (this.isHold && this.isNomination && this.isStart) {
            this.gestureDispatcher.source(this.presentId, this.startId);
        }
        this.startId = -1;
        this.isHold = false;
        this.isStart = false;
        this.gestureRecognizer.removeOrigin();
    }

    public void hitButton() {
        // 既に Nomination していた場合
        if (this.isNomination) {
            this.gestureDispatcher.change(this.presentId);
        }
    }
}

/*
 * 移動軌跡解析
 *
 */
package jp.kmsf.sui.controller.gesture;

import java.awt.Point;

/**
 * @author Shunsaku Yamazaki(sunsaku)
 */
public class GestureRecognizer {

    // 定数
    // NEXT と PREVIOUS の値、GestureHandler から参照される
    public static int NEXT = 1;
    public static int PREVIOUS = 2;
    private final int LEVEL = 36;
    private final int HOLE = 30;

    // private な重要な変数
    private Point origin;
    private Point center;

    // private な各メソッドで必要な変数
    private double exRadian = -1;
    private int popArea = -1;
    private int preArea = -1;
    private int exArea = -1;
    // [0] 右下右、[1] 下下右、[2] 下下左、[3] 左下左、[4] 左上左、[5] 上上左、[6] 上上右、[7] 右上右
    private int[] area = new int[8];


    public GestureRecognizer() {
    }

    // はじまり
    public void setOrigin(Point origin) {
        this.origin = origin;
    }

    // おわり
    public void removeOrigin() {
        this.origin = null;
        this.center = null;
        this.exRadian = -1;
        this.popArea = -1;
        this.exArea = -1;
        this.area = new int[8];
    }

    public int analogize(Point coordinate) throws GestureNotRecognizedException {
        // まだ中心が算出されてなければ、中心を算出する
        if (this.center == null) {
            this.center = analogizeCenter(coordinate);
        }
```

```
    // 中心角を算出する
    double radian = Math.atan2(
        (coordinate.getY() - this.center.getY()),
        (coordinate.getX() - this.center.getX()));
    if (radian < 0) {
        radian += Math.PI * 2;
    }

    // これが初めてでない
    if (0 < this.exRadian) {
        // 前の coordinate の中心角と比較して、360/LEVEL 移動しているかどうか
        double diff = this.exRadian - radian;
        this.exRadian = radian;
        if ((Math.PI / this.LEVEL) < Math.abs(diff)) {
            // 右回り
            if (0 < diff) {
                // 0 と 6 の境目の可能性が高いので左回りと見なす
                if (5 < Math.abs(diff)) {
                    return GestureRecognizer.NEXT;
                }
                return GestureRecognizer.PREVIOUS;
            }
            // 左回り
            else {
                // 0 と 6 の境目の可能性が高いので右回りと見なす
                if (5 < Math.abs(diff)) {
                    return GestureRecognizer.PREVIOUS;
                }
                return GestureRecognizer.NEXT;
            }
        }
    }
    this.exRadian = radian;

    throw new GestureNotRecognizedException();
}

// 中心を出す
// analogize から呼ばれる
private Point analogizeCenter(Point coordinate) throws GestureNotRecognizedException {
    // エリアをカウントする
    countArea(coordinate);

    // 中心に近ければ無視する
    if (Math.pow((coordinate.getX() - this.origin.getX()), 2) + Math.pow((coordinate.getY() - this.origin.getY()), 2)
        < Math.pow(this.HOLE, 2)) {
        throw new GestureNotRecognizedException();
    }

    // 前の coordinate が今の coordinate のエリアと違えば、線をまたいだことになるので中心を算出する
    if (this.preArea != this.exArea) {
        double x;
        double y;
        switch (this.preArea) {
            // [0] 右下右、[1] 下下右、[2] 下下左、[3] 左下左、[4] 左上左、[5] 上上左、[6] 上上右、[7] 右上右
            case 0:
                // [0] 右下右に [7] 右上右から進入してきた
                if (this.popArea == 7) {
                    x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                    y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
                    return new Point((int)x, (int)y);
                }
                // [0] 右下右に [1] 下下右から進入してきた
                else if (this.popArea == 1) {
                    x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                    y = this.origin.getY();
                    return new Point((int)x, (int)y);
                }
                break;
            case 1:
                if (this.popArea == 0) {
                    x = this.origin.getX();
                    y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
                    return new Point((int)x, (int)y);
                }
                else if (this.popArea == 2) {
                    x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                    y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
```

```
                return new Point((int)x, (int)y);
            }
            break;
        case 2:
            if (this.popArea == 1) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 3) {
                x = this.origin.getX();
                y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            break;
        case 3:
            if (this.popArea == 2) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY();
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 4) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() + (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            break;
        case 4:
            if (this.popArea == 3) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 5) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY();
                return new Point((int)x, (int)y);
            }
            break;
        case 5:
            if (this.popArea == 4) {
                x = this.origin.getX();
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 6) {
                x = this.origin.getX() - (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            break;
        case 6:
            if (this.popArea == 5) {
                x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 7) {
                x = this.origin.getX();
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            break;
        case 7:
            if (this.popArea == 6) {
                x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                y = this.origin.getY();
                return new Point((int)x, (int)y);
            }
            else if (this.popArea == 0) {
                x = this.origin.getX() + (this.origin.distance(coordinate) / 2);
                y = this.origin.getY() - (this.origin.distance(coordinate) / 2);
                return new Point((int)x, (int)y);
            }
            break;
        default:
            break;
    }
}
```

```
            throw new GestureNotRecognizedException();
        }

        // はいったエリアをカウントする
        // analogizeCenter から呼ばれる
        private void countArea(Point coordinate) {
            // まず、origin（始点）を中心として中心角を算出する
            double radian = Math.atan2((coordinate.getY() - this.origin.getY()), (coordinate.getX() - this.origin.getX()));
            if (radian < 0) {
                radian += Math.PI * 2;
            }

            // 次に、coordinate の存在するエリアを算出する
            // [0] 右下右、[1] 下下右、[2] 下下左、[3] 左下左、[4] 左上左、[5] 上上左、[6] 上上右、[7] 右上右
            for (int i = 0; i < 8; i++) {
                if (radian < Math.PI * 2 * (i + 1) / 8) {
                    this.area[i]++;
                    this.exArea = this.preArea;
                    this.preArea = i;
                    break;
                }
            }

            //  最多通行エリアを算出
            int max = this.area[0];
            for (int i = 1; i < 8; i++) {
                if (max < this.area[i]) {
                    max = this.area[i];
                    this.popArea = i;
                }
            }
        }
    }
```