

平成 27 年度卒業論文

---

**FingCV: スマートフォンの内蔵カメラを用いたジェスチャによる  
簡易操作法の提案・実装・評価**

---

電気通信大学 情報理工学部  
情報・通信工学科 コンピュータサイエンスコース

赤池 HI\* 研究室

指導教員：赤池 英夫 (*Akaike Hideo*)

学籍番号：1211019 / 猪膝 孝之 (*Inohiza Takayuki*)

提出日：平成 28 年 2 月 1 日 (月)

---

\*Human Interface

## 概要

本研究では、スマートフォンの内蔵カメラを用いて指のジェスチャを認識して操作をする方法について提案、実装し、有用性について評価した。

近年スマートフォンが普及しており、タッチ操作が可能になった事によって、ユーザーは従来の携帯電話よりも直感的な操作を行う事が出来るようになった。加えて、スマートフォンは以前よりも画面表示領域が大きくなっている。しかし、ユーザーがコンテンツを視認する際に手や指が邪魔になり、スマートフォンの表示領域の広さを最大限生かせなくなる。

そこで本研究はスマートフォンの内蔵カメラを利用して指のジェスチャを認識することによって、表示領域の広さを生かすような操作方法を提案し、実装した。また、独自のテスト用ページを作成し、被験者のブラウジング実験により有用性について評価した。

実験結果より FingCV のほうがセッション数を増やせば増やすほどタスク完了時間が低下することがわかった。操作に慣れない人も多く、スクロールやズームの精度がそこまで良くなかったため、被験者 8 人中良いと思った人が 1 人、普通と思った人が 4 人、悪いと思った人が 3 人と FingCV の総合的な評価としてはあまり良くなかった。個人差もかなりあったように見られた。画面の表示領域を最大限生かすという点は、FingCV の場合スクロールやズームをしている際、画面全体の把握がしやすかったどうかという設問で、とても把握しやすいと思った人が 2 人、把握しやすいと思った人が 3 人と回答しており、改善されたという意見が多くなったことがわかった。しかし、カメラのプレビュー画面に集中しすぎて画面全体が見づらくなってしまうという問題点も挙がった。

# 目次

<b>1</b>	<b>はじめに</b>	<b>5</b>
1.1	研究背景	5
1.2	研究目的	5
<b>2</b>	<b>関連研究</b>	<b>6</b>
2.1	はじめに	6
2.2	背面から端末を操作	6
2.3	内蔵カメラを用いて端末を操作	6
2.4	おわりに	7
<b>3</b>	<b>本システム (FingCV) の概要</b>	<b>8</b>
3.1	はじめに	8
3.2	主な仕様、操作方法	8
3.3	スワイプ操作	9
3.4	ピンチイン、ピンチアウト操作	11
3.5	閲覧履歴の移動操作	12
3.6	おわりに	13
<b>4</b>	<b>実装</b>	<b>14</b>
4.1	はじめに	14
4.2	実装環境	14
4.3	指認識	14
4.3.1	肌色領域の抽出	14
4.3.2	手の検出	15
4.3.3	指先の検出	16
4.4	ブラウザの操作処理	20
4.4.1	操作モードの切替	20
4.4.2	指先のジェスチャ検出	21
4.5	おわりに	22
<b>5</b>	<b>評価実験</b>	<b>23</b>
5.1	はじめに	23
5.2	実験目的	23
5.3	被験者情報	23
5.4	実験内容	23
5.5	アンケート内容	24
5.6	おわりに	24
<b>6</b>	<b>実験の結果と考察</b>	<b>25</b>
6.1	はじめに	25
6.2	測定結果	25
6.3	アンケート結果	27
6.3.1	スクロールの精度	27
6.3.2	ズームの精度	27

6.3.3	疲労感の有無	28
6.3.4	スクロールやズームをしている際の画面全体の把握のしやすさ(タッチの場合)	28
6.3.5	スクロールやズームをしている際の画面全体の把握のしやすさ(FingCVの場合)	28
6.3.6	タッチ操作とFingCVの比較	28
6.3.7	総合的なFingCVの評価	29
6.3.8	追加してほしい機能	29
6.3.9	その他	29
6.4	おわりに	30
<b>7</b>	<b>おわりに</b>	<b>31</b>
7.1	まとめ	31
7.2	改善すべき点	31
7.3	展望	31

## 図目次

1	背面から端末を操作	6
2	内蔵カメラを用いて端末を操作	7
3	操作方法	8
4	画面の端の特定の位置	8
5	動作時の画面	9
6	初期位置	9
7	画面の左下の位置	10
8	スワイプ操作	10
9	画面の中央下の位置	11
10	ピンチイン、ピンチアウト操作	12
11	画面の右下の位置	12
12	閲覧履歴の移動操作	13
13	指先の検出	14
14	2値化の例	15
15	距離変換処理	16
16	手の検出	16
17	肌色領域の重心	17
18	内積によるなす角度	17
19	指と判断された多数の点	19
20	2つの点を1つの点にする処理	19
21	残った最後の1点	20
22	ブラウザの操作処理の概要	20
23	初期位置として赤い円を描写	21
24	実験時の画面	24
25	FingCVで行った場合のタスク完了時間	25
26	タッチ操作で行った場合のタスク完了時間	26

## 表 目 次

1	被験者情報	23
2	ユーザーごとのタスク完了時間(FingCVの場合)	25
3	ユーザーごとのタスク完了時間の平均(FingCVの場合)	26
4	ユーザーごとのタスク完了時間(タッチ操作の場合)	26
5	ユーザーごとのタスク完了時間の平均(タッチ操作の場合)	27
6	スクロールの精度はどうでしたか？	27
7	ズームの精度はどうでしたか？	28
8	疲労感はありましたか？	28
9	タッチ操作の場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？	28
10	本システムの場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？	28
11	タッチ操作と比べて本システムはどうでしたか？	29
12	総合的に見て本システムはどうでしたか？	29

# 1 はじめに

## 1.1 研究背景

近年、従来の携帯電話の代わりにスマートフォンの急速な普及が見られる。スマートフォンは、従来の携帯電話に比べて画面の表示領域が広いことに加えて、タッチして操作する事が可能となっている点が特徴である。

タッチ操作が可能になった事によって、ユーザーは従来の携帯電話よりも直観的な操作を行う事ができるようになった。しかし、コンテンツを視認する際にタッチ操作では手や指が邪魔になり、スマートフォンの表示領域の広さを最大限生かせなくなる。

画面の表示領域を最大限生かすために端末の背面に背面タッチが可能になるようなハードウェアを取り付けることによって背面からの端末の操作にする研究 [1][2] が行われてきた。しかし、欠点として可搬性が制限されてしまったり、独自のハードウェアとして専用の部品を取り付ける必要がある。

他には内蔵カメラを用いて指を認識しジェスチャを検出することによって端末を操作を可能にする研究 [3][4] が行われてきた。しかし、欠点としてスワイプ操作しかできないので機能性が欠ける。

そこで、独自のハードウェアを装着することなく、機能性を向上させることができないかと考え、本研究の立案に至った。

## 1.2 研究目的

本研究は、どの端末でも付属している内蔵カメラを用いて画面の表示領域を狭める事なく、スマートフォンを操作するような手法を提案、実装を行った。その後、評価実験を行い、有用性について評価を行った。

## 2 関連研究

### 2.1 はじめに

この章では、画面の表示領域を最大限生かすことを目的とした既存の研究についてそれぞれ特徴や問題点を述べる。

### 2.2 背面から端末を操作

Wigdor ら [1] の研究では図 1a のようにディスプレイの背面にタッチパッドとカメラを装着し、背面からの操作を可能にしている。しかし、背面にカメラを取り付けるために可搬性が制限されてしまうという問題がある。

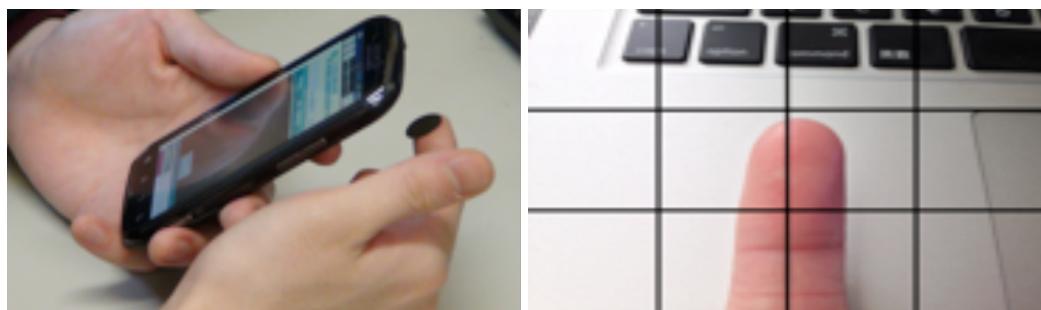
小林ら [2] の研究では図 1b のように端末の背面にタッチパネルを取り付けて画面領域を狭める事なく操作をする方法を可能としている。しかし、独自のハードウェアを装着する必要があることが問題として考えられる。



図 1: 背面から端末を操作

### 2.3 内蔵カメラを用いて端末を操作

竹田ら [3] の研究では図 2a のように端末の背面カメラで指を認識し上下左右の 4 方向のジェスチャを検出することによって操作する手法を実現した。また、それを改良した研究 [4] では図 2b より斜め移動が加わり、より自由度の高いジェスチャの検出することが可能となっている。しかし、この研究は検討段階であり、実際に実装されたわけではない。加えて、使用できるジェスチャの数が限定されてしまい、スワイプ操作しかできないので機能性が欠けることが考えられる。



(a) 上下左右の 4 方向のジェスチャを検出

(b) 斜め移動のジェスチャの検出が可能

図 2: 内蔵カメラを用いて端末を操作

## 2.4 おわりに

本章では、画面の表示領域を最大限生かすことを目的とした既存の研究について述べた。次章では、本研究で作成したシステム、FingCV の概要について述べる。

### 3 本システム (FingCV) の概要

#### 3.1 はじめに

この章では、本研究で作成したシステム、FingCVについて詳しく述べる。最初に、システムの主な仕様、操作方法について説明をし、その次に実装したそれぞれの機能について述べる。

#### 3.2 主な仕様、操作方法

ここではブラウザ利用時を想定する。右利きの場合、図3のように左手でスマートフォンを持ち、右手の指を背面カメラで読み取って、指の動きを認識することによってジェスチャを検出する。



図3: 操作方法

図4のような画面の端の特定の位置を押したままにすることによって操作モードが切り替わるようにする。操作モードは以下の3つである。

- スワイプ操作
- ピンチイン、ピンチアウト操作
- 閲覧履歴の移動操作

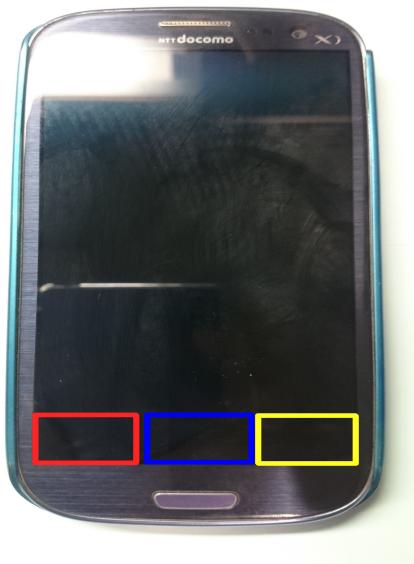


図4: 画面の端の特定の位置

図5のように右利き用には画面右下でカメラのプレビューを見ることができるようになっており、指がカメラに写っているかどうかを確認することができる。

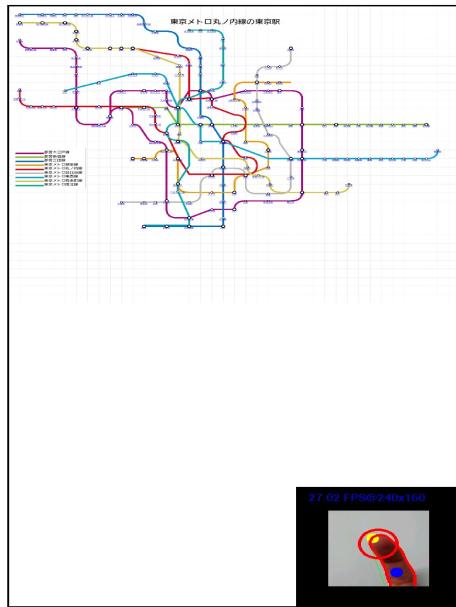


図5: 動作時の画面

図6はカメラのプレビュー画面である。この画面には赤い丸い円が描画されており、初期位置としてこの円の中に指先を写すようとする。

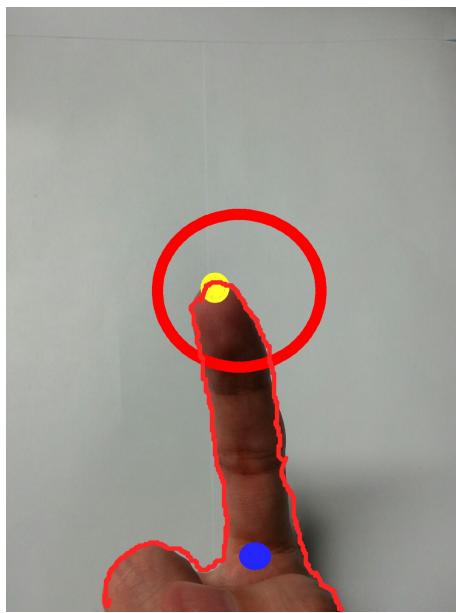


図6: 初期位置

### 3.3 スワイプ操作

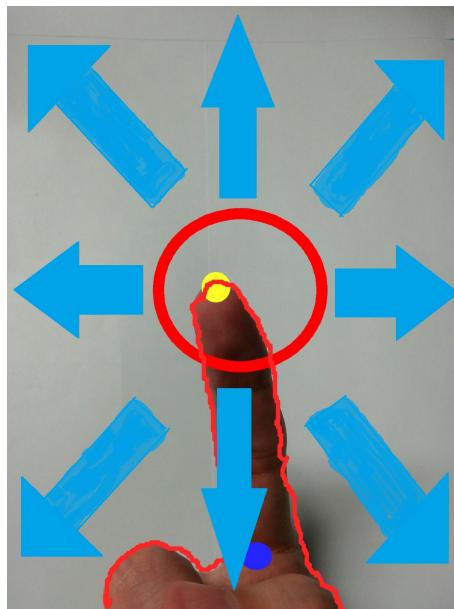
右利き用の場合、図7のように画面の左下の位置を指で押したままにしている時、スワイプ操作モードに移行する(左利き用は画面の右下の位置)。スワイプとは画面に触れた状態で指を滑らせる動作のこ

とである。



図 7: 画面の左下の位置

まず、初期位置である赤い丸い円の中に指先を写すようする。円の中から外に向かって指先を移動させ、指先が円の外に出た際にジェスチャを検出する。図 8 のように 8 方向の移動が可能である。タッチ操作のスワイプと同様の方向に動かすようする。例えば、指先を円の中から左方向に移動させると画面は右方向にスクロールをし、円の中から下方向に移動させると画面は上方向にスクロールする。



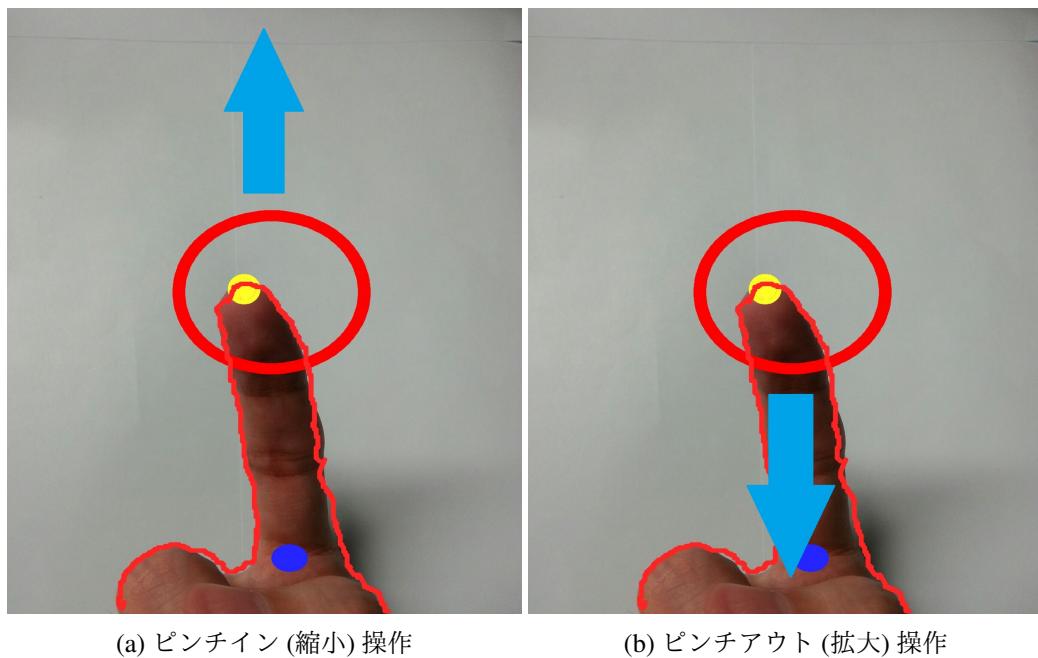
### 3.4 ピンチイン、ピンチアウト操作

図9のように画面の中央下の位置を指で押したままにしている時、ピンチイン、ピンチアウト操作モードに移行する。ピンチインとは二本の指を画面上にのせてその間隔を縮める動作のことと、反対にピンチアウトはその間隔を広げる動作のことである。



図9: 画面の中央下の位置

まず、初期位置である赤い丸い円の中に指先を写すようにする。図10aのように指先を円の中から上方向に移動させるとピンチイン(縮小)操作をし、図10bのように円の中から下方向に移動させるとピンチアウト(拡大)操作をする。



(a) ピンチイン(縮小)操作

(b) ピンチアウト(拡大)操作

図 10: ピンチイン、ピンチアウト操作

### 3.5 閲覧履歴の移動操作

閲覧履歴とはブラウザの履歴のことであり、履歴がある場合は進んだり戻ったりすることができる。右利き用の場合、図 11 のように画面の右下の位置を指で押し続けている時、閲覧履歴の移動操作モードに移行する(左利き用は画面の左下の位置)。



(a) 画面の端(右下)

(b) 右下の位置を長押し

図 11: 画面の右下の位置

まず、初期位置で赤い丸い円の中に指先を写すようにする。図 12a のように指先を円の中から右方向

に移動させると閲覧履歴を一つ進むことができ、図 12b のように円の中から左方向に移動させると閲覧履歴を一つ戻ることができる。履歴がない場合はページは変化しない。

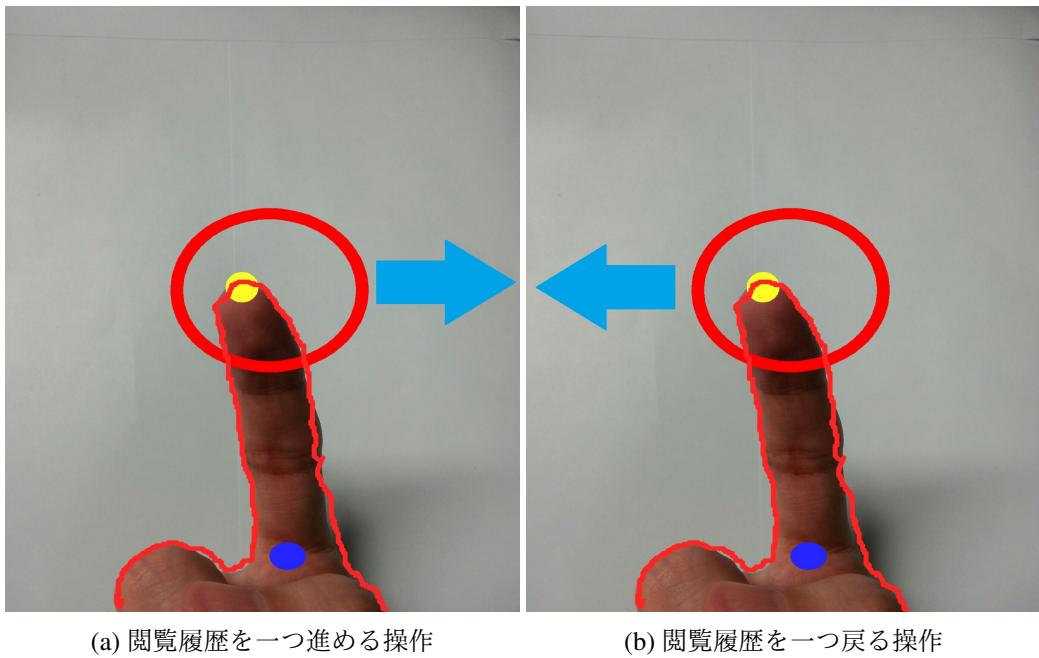


図 12: 閲覧履歴の移動操作

### 3.6 おわりに

本章では本研究で作成したシステム、FingCV についての主な仕様、操作方法について説明をし、その後に実装したそれぞれの機能について述べた。次章では、FingCV の実装環境や実装方法について詳しく述べる。

## 4 実装

### 4.1 はじめに

この章では最初に本研究で作成したシステム、FingCVについての実装環境について説明をする。その次に指認識の方法について詳しく説明した後にブラウザの操作処理について詳しく述べる。

### 4.2 実装環境

システムにはプログラミング言語として Java を使い、Android 上に作成した。Android 端末は GALAXY S3(SC-06D) を使用した。指認識には画像処理ライブラリの OpenCV for Android を使用し、Web ページの表示には WebView を使用した。

### 4.3 指認識

ここではブラウザの操作をするため作成した指の検出の概要と流れを述べる。指の検出は画像処理によって行い、流れは図 13 のようになっている。既存の研究 [5] の手法を参考にして指認識の手法を考案した。また、本研究では画像処理による指先の検出を実現するために OpenCV for Android を使用した。

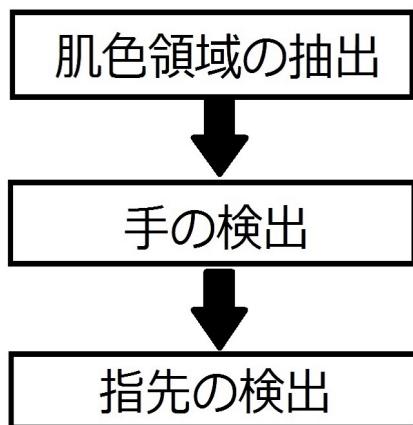


図 13: 指先の検出

指先の検出には、最初にカメラの画面内に表示される肌色領域を抽出する。次に肌色領域の最大のところを手と判断し、ラベリングをし輪郭線を取得する。その後に指先を検出する。それぞれの詳細については以降に述べる。

#### 4.3.1 肌色領域の抽出

指先を検出するために、まずカメラから入力された画像から肌色にあたる領域を抽出する。一般的な肌色領域の抽出方法としてはカメラによって入力された RGB 表色系の画像を他の画像に変換した後、画像の 2 値化という作業を行う。画像の 2 値化とは、任意の方法で検出したい色の閾値を決め、その閾値を元に画像を二色に分ける作業である。2 値化の例を図 14 に示す。肌色の 2 値化としては、肌色にあたる色の閾値を決め、その閾値から 2 値化をし、肌色であるか否かを判断する。また、RGB 表色系の画像を用いない理由は、RGB 表色系では R・G・B のような単色の検出率は非常に高いが、肌色のような

混合色を抽出する場合、RGB 表色系では混合色は広範囲に分布するため、閾値を決める方法で肌色領域を抽出する事は誤検出が非常に多くなるからである。RGB 表色系を変換する表色系に HSV 表色系を用いた。HSV 表色系とは、1978 年にアルヴィ・レイ・スミス (Alvy Ray Smith) によって考案された表色系であり、色相 (Hue)、彩度 (Saturation · Chroma)、明度 (Brightness · Lightness · Value) の三つの成分からなる表色系であり、これは RGB 色空間の非線形空間である。HSV 表色系を用いる理由としては肌色の分布する範囲が狭いという特徴があるからである。肌色を抽出するための HSV 表色系での H · S · V のそれぞれの閾値を以下に示す。

- Hue:  $0 < H < 25$
- Saturation:  $58 < S < 173$
- Value:  $88 < V < 229$

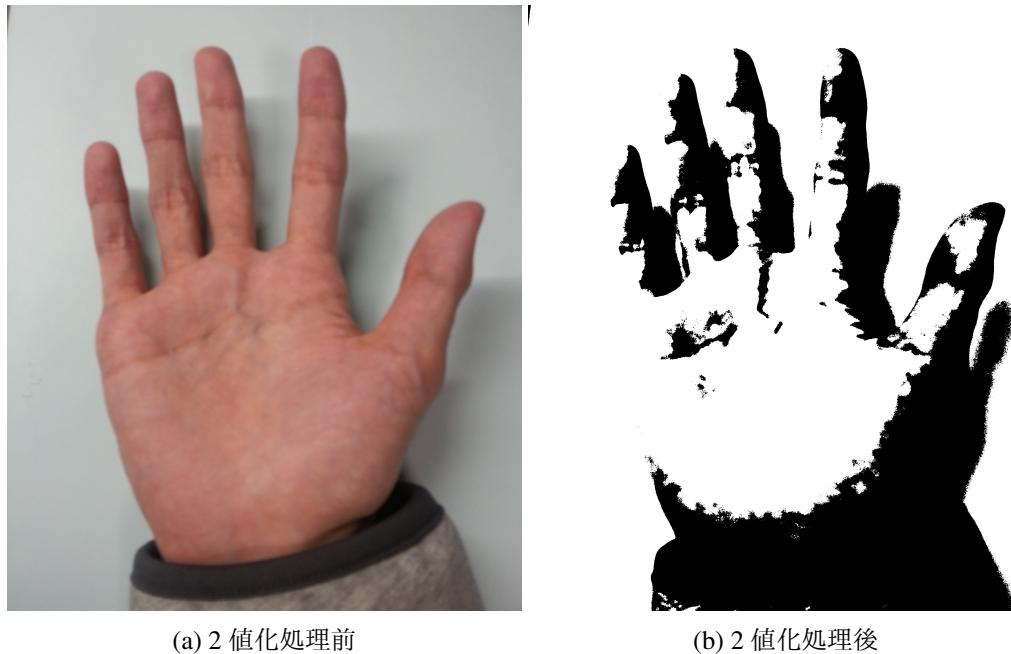
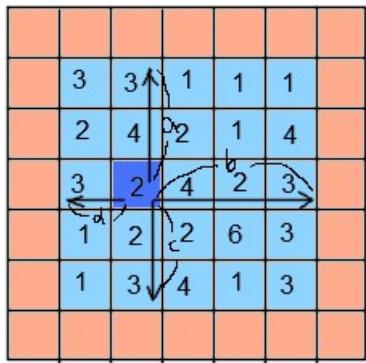


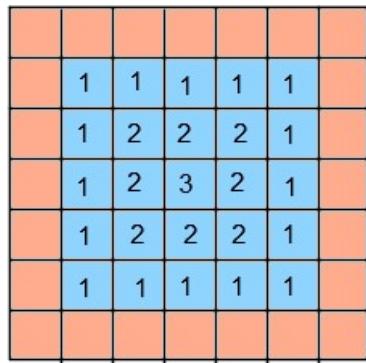
図 14: 2 値化の例

#### 4.3.2 手の検出

次に目的の指先を検出するために手を検出する。手を検出する際に留意点としては、手以外で肌色に近い色の物体が存在する可能性を考慮しなければならない。本研究のシステムではカメラから入力される画像中の最大面積の肌色の物体を手とした。処理の軽減のため距離変換画像を用いた。距離変換とは二値画像の中にいくつか存在する物体の画素値(画素の明るさを表す数値)を、その物体の任意の画素から背景画素(画素値が 0)までの最短距離を求め、画素値を置き換える処理である。変換して得られた画像を距離変換画像という。図 15 に距離変換処理の例をあげる。



(a) 距離変換前



(b) 距離変換後

図 15: 距離変換処理

上の方法を用いて手を検出する方法をまとめると、手はカメラからの画像中最大の面積を持つ肌色領域としているので、背景から最も遠い画素の画素値が画像中最大になる事になる。つまり、距離変換画像中の最大の画素値を持つ肌色領域が手という事になる。

図 16 が手を検出した例である。



図 16: 手の検出

#### 4.3.3 指先の検出

次に指先の検出方法について述べる。まず、最初に図 17 のように先ほど抽出された肌色領域の重心を求めて、その重心より上を指と判断するようとする。

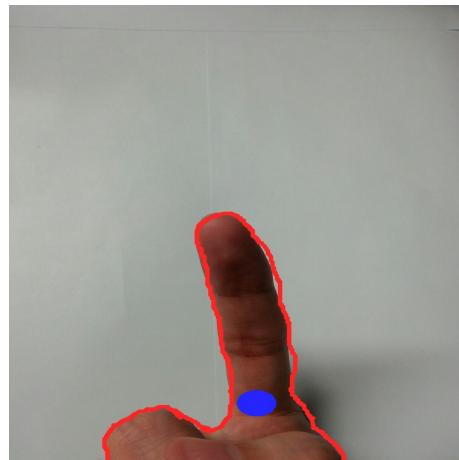


図 17: 肌色領域の重心

次に指という形状を生かして輪郭の凹凸の有無と凹凸の方向から判定を行う。先ほど抽出された肌色領域の輪郭の輪郭点を一定の間隔をあけ 3 点  $P_1, P_2, P_3$  で、始まりから終わりまで沿うようにして調べていく。調べる内容は画像中の座標、x、y 座標であり、この 3 点  $P_1, P_2, P_3$  の座標情報から  $P_2$  を原点とし、図 18 のようになす角度  $\theta$  を計算する。なす角度  $\theta$  が式 1 の際、その 3 点から作られる形状は凹凸の形状をしている判断する。なす角度の計算方法は  $P_2$  を原点とした  $P_1, P_2, P_3$  の内積の計算式 18 から算出する。

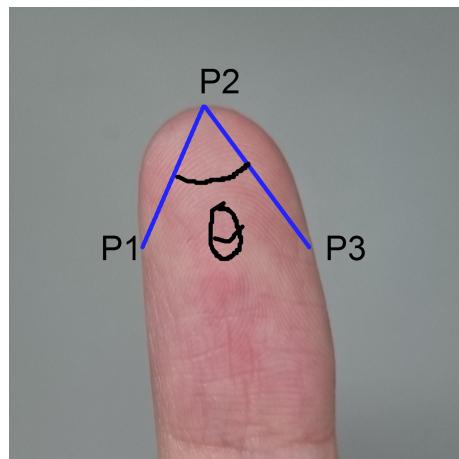


図 18: 内積によるなす角度

$$0^\circ < \theta < 60^\circ \quad (1)$$

$$P_1 = (P_{1x}, P_{1y}) \quad (2)$$

$$P_2 = (P_{2x}, P_{2y}) \quad (3)$$

$$P_3 = (P_{3x}, P_{3y}) \quad (4)$$

$$C_1 = (C_{1x}, C_{1y}) \quad (5)$$

$$C_{1x} = P_{1x} - P_{2x} \quad (6)$$

$$C_{1y} = P_{1y} - P_{2y} \quad (7)$$

$$C_2 = (C_{2x}, C_{2y}) \quad (8)$$

$$C_{2x} = P_{3x} - P_{2x} \quad (9)$$

$$C_{2y} = P_{3y} - P_{2y} \quad (10)$$

$$P_1 P_2 = C_1 \quad (11)$$

$$P_1 P_2 = C_2 \quad (12)$$

$$|P_1 P_2| = |C_1| \quad (13)$$

$$|P_3 P_2| = |C_2| \quad (14)$$

$$|C_1| = \sqrt{|C_{1x}|^2 + |C_{2y}|^2} \quad (15)$$

$$|C_2| = \sqrt{|C_{2x}|^2 + |C_{2y}|^2} \quad (16)$$

$$C_1 \cdot C_2 = C_{1x} \times C_{2x} + C_{1y} \times C_{2y} \quad (17)$$

$$\cos \theta = \frac{C_1 \cdot C_2}{|C_1||C_2|} \quad (18)$$

次に凹凸の方向の判定を行う。凹凸の方向の判定には  $P_2$  を原点とした  $P_1, P_2, P_3$  の外積の式から算出される  $\sin \theta$  を使用した。本研究では  $\sin \theta$  が式 19 の時に凸と判定し、式 20 の時に凹と判定した。外積の計算には式 22 を使用した。

$$0^\circ < \theta < 90^\circ \quad (19)$$

$$-90^\circ < \theta < 0^\circ \quad (20)$$

$$C_1 \times C_2 = C_{1x} \times C_{2y} - C_{2x} \times C_{1y} \quad (21)$$

$$\sin \theta = \frac{C_1 \times C_2}{|C_1||C_2|} \quad (22)$$

次に人間の指は特殊な姿勢を除いては、基本的に上に向いているので凸の方向に向いている 3 点  $P_1, P_2, P_3$  を指と判定した。ここで先ほどの指先の検出の判定をまとめると内積から得られるなす角度  $\theta$  が式 1 の場合、かつ、外積から得られるなす角度が式 19 の場合、その 3 点は指だと判定している。ここまで処理で指の判断は可能になったが、指と判断される 3 点は輪郭点の調査を手の輪郭点に対して全てに行う事から、図 19 のように 1 つの指に対して多数存在する事になる。

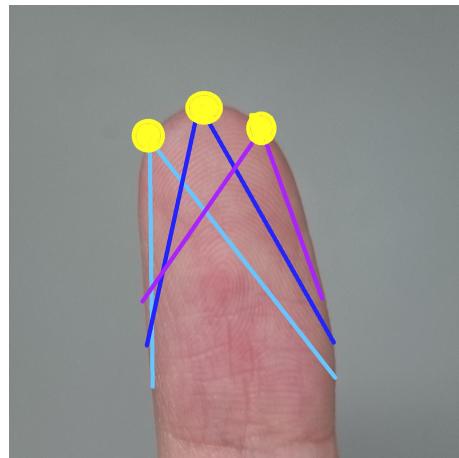


図 19: 指と判断された多数の点

次に先ほど多数検出された点の中で、ある 2 つの点が近い距離にある場合、それらを 1 つにする処理をする。図 20 のようにそれぞれの点の x,y 座標を足して 2 で割って 1 つにする処理をする。

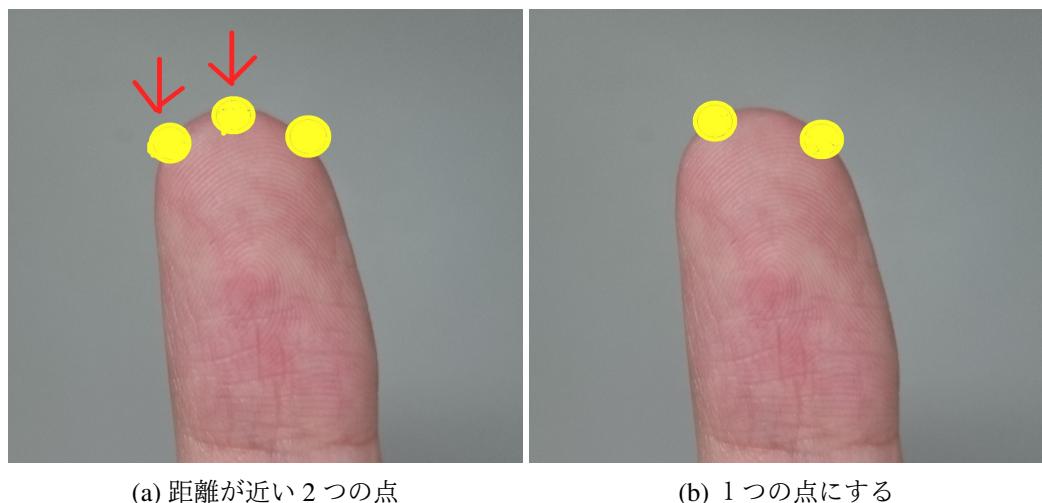


図 20: 2 つの点を 1 つの点にする処理

同様に検出された多数の点が 1 つの点になるまで繰り返し処理をする。図 21 のように残った最後の 1 点を指先と判断する。

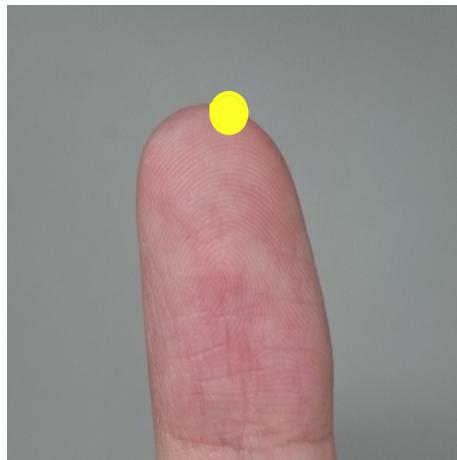


図 21: 残った最後の 1 点

#### 4.4 ブラウザの操作処理

ここではブラウザの操作処理について詳しく説明をする。Web ページの表示には WebView を使用する。右利きの場合、左手の親指で画面の特定の位置を長押しすることによって操作モード切り替える。以降は、操作モードの切り替え、指先のジェスチャ検出の処理方法について述べる。

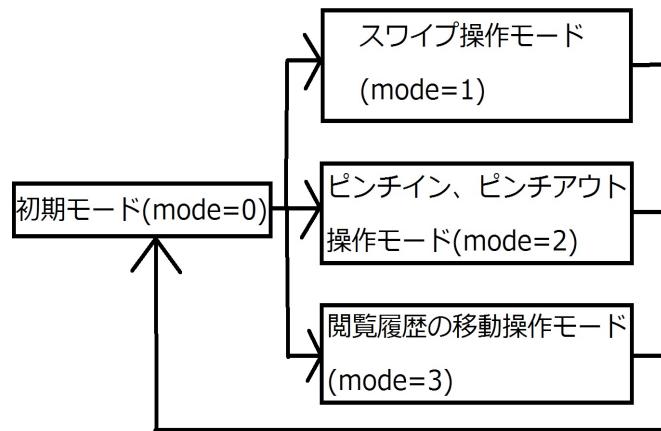


図 22: ブラウザの操作処理の概要

##### 4.4.1 操作モードの切替

右利きの場合、左手の親指で画面の特定の位置を長押しすることによって操作モード切り替える。どこも押していない場合は初期モード (mode=0) となっている。画面のサイズは縦が 1280 で、横が 720 で、左上を原点とする。画面を押している場所の座標を (pointX, pointY) として保存する。その時、操作モードの切替処理は以下のようになる。

- $0 \leq \text{pointX} \leq 240$ かつ $1120 \leq \text{pointY} \leq 1280$ のとき、mode=1(スワイプ操作モード)
- $241 \leq \text{pointX} \leq 480$ かつ $1120 \leq \text{pointY} \leq 1280$ のとき、mode=2(ピンチイン、ピンチアウト操作モード)
- $481 \leq \text{pointX} \leq 720$ かつ $1120 \leq \text{pointY} \leq 1280$ のとき、mode=3(閲覧履歴の移動操作モード)

#### 4.4.2 指先のジェスチャ検出

まず、図 23 のようにカメラのプレビュー画面に初期位置として赤い丸い円を描画する。この赤い円の中に指先(黄色の円)を写すとその座標を(yubi1.x, yubi1.y)として保存する。

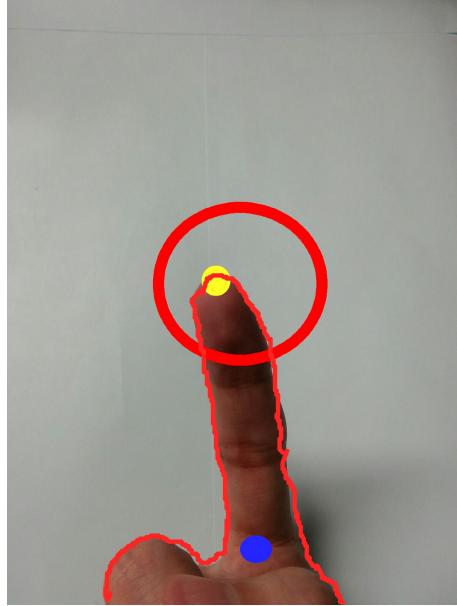


図 23: 初期位置として赤い円を描写

指先が赤い円の外で検出された場合、Timer、TimerTask を利用する。Timer クラスの schedule メソッドを使用し、指定した時間に指定したタスクが実行されるようにスケジュールをする。本システムの場合は指先が赤い円の外で検出されて 0.5 秒にタスクが実行されるようにスケジュールをする。

タスクの内容はまず指先の座標を(yubi2.x, yubi2.y)として保存する。そして、yubi1.x と yubi2.x, yubi1.y と yubi2.y のそれぞれの差を取りて一定の大きさ以上だった場合、指先のジェスチャを検出するようにする。

まず、スワイプ操作について詳しく説明する。WebView クラスの scrollBy メソッドを使用する。8 方向のジェスチャの処理については以下のようにする。N は 15 とし、shokika() では yubi2.x=yubi1.x、yubi2.y=yubi1.y という処理をしている。

```
if (yubi2.x - yubi1.x > N && yubi2.y - yubi1.y <= N && yubi1.y - yubi2.y <= N) {  
    if (@< webView.getScrollY()) {  
        webView.scrollBy(@, -50); // 上方向のスクロール  
    }  
    shokika();  
} else if (yubi1.x - yubi2.x > N && yubi2.y - yubi1.y <= N && yubi1.y - yubi2.y <= N) {  
    if (webView.getScrollY() < webView.getContentHeight() * webView.getScale()) {  
        webView.scrollBy(@, 50); // 下方向のスクロール  
    }  
    shokika();  
} else if (yubi2.y - yubi1.y > N && yubi2.x - yubi1.x <= N && yubi1.x - yubi2.x <= N) {  
    if (webView.getScrollX() < 600 * webView.getScale()) {  
        webView.scrollBy(50, 0); // 右方向のスクロール  
    }  
    shokika();  
} else if (yubi1.y - yubi2.y > N && yubi2.x - yubi1.x <= N && yubi1.x - yubi2.x <= N) {  
    if (@< webView.getScrollX()) {  
        webView.scrollBy(-50, 0); // 左方向のスクロール  
    }  
}
```

```

        shokika();
    } else if (yubi1.x - yubi2.x > N && yubi1.y - yubi2.y > N) {
        if (webView.getScrollY() < webView.getContentHeight() * webView.getScale()
            && 0 < webView.getScrollX()) {
            webView.scrollBy(0, 50); // 左下
            webView.scrollBy(-50, 0);
        }
        shokika();
    } else if (yubi1.x - yubi2.x > N && yubi2.y - yubi1.y > N) {
        if (webView.getScrollY() < webView.getContentHeight() * webView.getScale()
            && webView.getScrollX() < 600 * webView.getScale()) {
            webView.scrollBy(0, 50); // 右下
            webView.scrollBy(50, 0);
        }
        shokika();
    } else if (yubi2.x - yubi1.x > N && yubi1.y - yubi2.y > N) {
        if (0 < webView.getScrollY() && 0 < webView.getScrollX()) {
            webView.scrollBy(0, -50); // 左上
            webView.scrollBy(-50, 0);
        }
        shokika();
    } else if (yubi2.x - yubi1.x > N && yubi2.y - yubi1.y > N) {
        if (0 < webView.getScrollY() && webView.getScrollX() < 600 * webView.getScale()) {
            webView.scrollBy(0, -50); // 右上
            webView.scrollBy(50, 0);
        }
        shokika();
    }
}

```

次にズームイン、ズームアウト操作について詳しく説明する。WebView クラスの zoomIn、zoomOut メソッドを使用する。それぞれの操作については以下のようにする。

```

if (yubi2.x - yubi1.x > 22) {
    webView.zoomIn(); // ズームイン
    shokika();
} else if (yubi1.x - yubi2.x > 22) {
    webView.zoomOut(); // ズームアウト
    shokika();
}

```

次に閲覧履歴の移動操作について詳しく説明する。WebView クラスの goForward、goBack メソッドを使用する。それぞれの操作について以下のようにする。

```

if (yubi1.y - yubi2.y > 25) {
    webView.goForward(); // 進む
    shokika();
} else if (yubi2.y - yubi1.y > 25) {
    webView.goBack(); // 戻る
    shokika();
}

```

## 4.5 おわりに

本章では本研究で作成したシステム、FingCV についての実装環境について説明をし、指認識、ブラウザの操作処理の方法について詳しく説明をした。次章では評価実験について述べる。

## 5 評価実験

### 5.1 はじめに

この章では本研究で作成したシステム、FingCVについての評価実験を行う。最初に実験目的について述べ、その後に実験内容について詳しく説明をする。

### 5.2 実験目的

本実験により、FingCVを使用することによってスマートフォンの表示領域の広さを最大限生かすことができたかどうかについて評価をする。被験者には実験後、アンケートに回答してもらい FingCV の使用感についても評価をする。

### 5.3 被験者情報

8人の被験者に本実験を依頼した。被験者全員の詳細なデータは下の表にまとめた。

被験者 No.	性別	年齢	利き手
01	男	21	右利き
02	男	23	右利き
03	男	22	右利き
04	男	22	右利き
05	男	23	左利き
06	男	24	右利き
07	女	20	右利き
08	男	24	右利き

表 1: 被験者情報

表 1 より、被験者の男女比は 7:1 で、年齢の中央値は 22.5 歳であった。全員学生であり、普段からスマートフォンを利用している。

### 5.4 実験内容

被験者に本システムである FingCV とタッチ操作の方法で、図 24 のような作成した独自のページについて実験を行った。作成した独自のページには jQuery のプラグイン [6][7] を使用している。被験者には路線図の中からある特定の駅を探すタスクを FingCV のほうで 10 回、タッチ操作の方法で 10 回それぞれ行った。路線図を開いた時、開始時間を測定するようにし、特定の駅が見つかり、その場所をタッチした時、終了時間を測定するようにし、終了時間から開始時間を引いたものをタスク完了時間として測定した。測定をする前に被験者には慣れるまでそれぞれの操作の練習をしてもらった。被験者 No. が奇数の人には最初に FingCV のほうで 10 回タスクを行った後に、タッチ操作の方法で 10 回行ってもらい、偶数の人には最初にタッチ操作の方法で 10 回タスク行った後に、FingCV のほうで 10 回行ってもらった。それぞれの正解の駅はランダムであり、池袋や上野のような利用者が多くて明らかに認知度が高い駅や一度出た駅は出ないようにした。

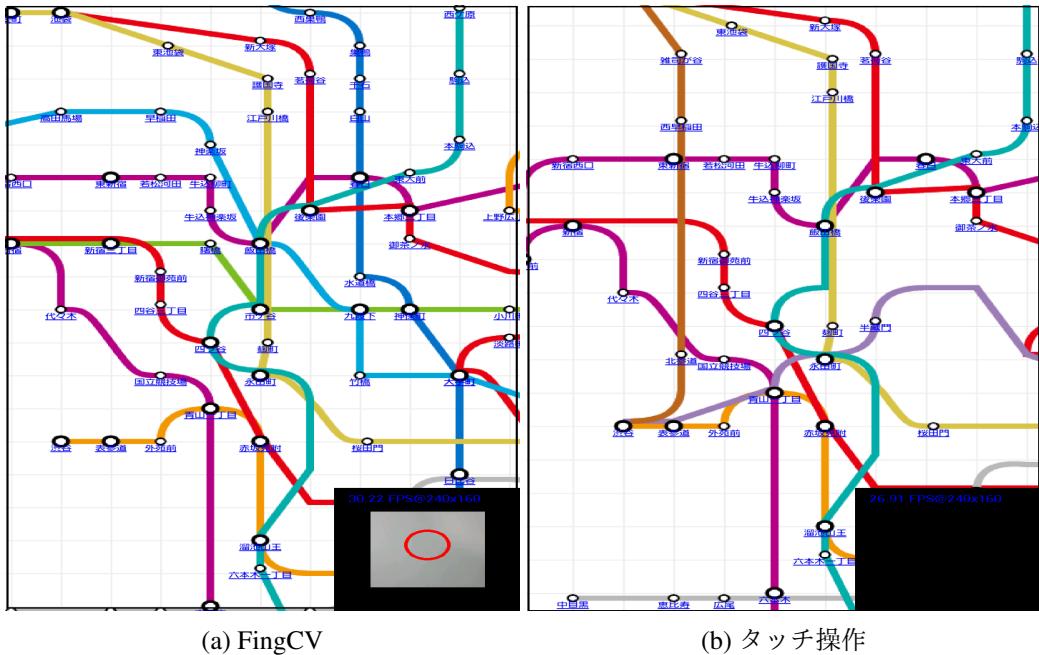


図 24: 実験時の画面

## 5.5 アンケート内容

実験後、被験者には FingCV の使用感についてアンケートで回答してもらった。アンケートの内容は以下である。

- スクロールの精度についてはどうでしたか？
- ズームの精度はどうでしたか？
- 疲労感はありましたか？
- タッチ操作の場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？
- 本システムの場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？
- タッチ操作と比べて本システムはどうでしたか？
- 総合的に見て本システムはどうでしたか？
- 何か追加してほしい機能等はありますか？
- 何か気づいたこと等あればご自由にどうぞ

## 5.6 おわりに

本章では本実験の目的、実験内容、アンケート内容について詳しく述べた。次章では実験で得られたタスク完了時間の結果やアンケート結果について詳しく述べる。

## 6 実験の結果と考察

### 6.1 はじめに

この章では本実験の測定結果やアンケート結果について詳しく述べる。それぞれの考察についても述べる。

### 6.2 測定結果

被験者 8 人の FingCV のほうのタスク完了時間は以下のようになった。

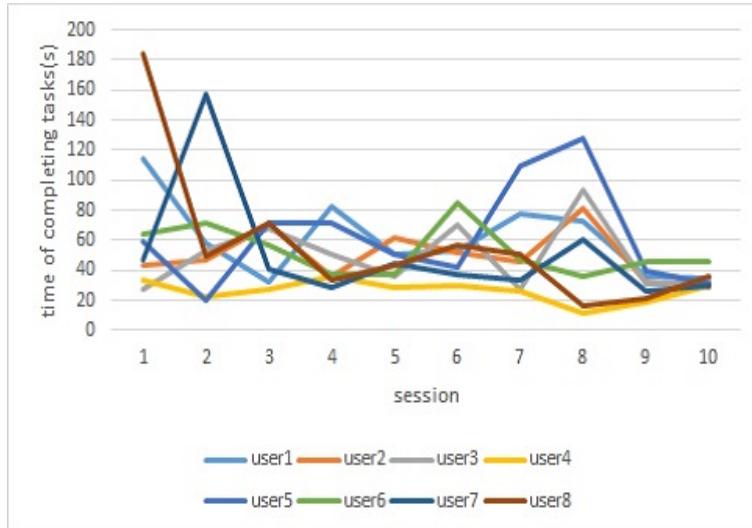


図 25: FingCV で行った場合のタスク完了時間

user	1	2	3	4	5	6	7	8	9	10
user 1	114.806	57.588	31.889	82.285	50.526	52.903	77.617	72.354	36.034	34.932
user 2	43.218	47.418	70.216	36.504	61.229	51.901	45.806	81.283	31.669	27.949
user 3	26.824	53.46	68.106	50.858	36.452	70.822	27.045	93.365	30.571	31.772
user 4	33.812	22.704	27.016	35.812	28.923	30.349	26.721	11.236	18.7	29.546
user 5	59.211	20.43	71.151	71.697	50.737	42.415	108.932	127.294	39.299	30.519
user 6	63.769	71.88	57.02	36.952	37.474	84.948	47.011	35.842	45.12	45.924
user 7	47.308	156.765	40.68	28.174	44.001	37.012	33.869	59.931	25.506	29.636
user 8	183.779	49.882	71.324	33.603	43.751	57.333	50.102	16.657	21.341	35.358

表 2: ユーザーごとのタスク完了時間 (FingCV の場合)

user	平均	平均 (6~10)	平均 (9~10)
user 1	61.0934	54.768	35.483
user 2	49.7703	47.7216	29.809
user 3	48.9274	50.715	31.1715
user 4	26.4819	23.3104	24.123
user 5	62.1685	69.6918	34.909
user 6	52.603	51.787	45.567
user 7	50.2882	37.1908	27.571
user 8	56.313	36.1582	28.3495

表 3: ユーザーごとのタスク完了時間の平均 (FingCV の場合)

タッチ操作のほうのタスク完了時間は以下のようになつた。

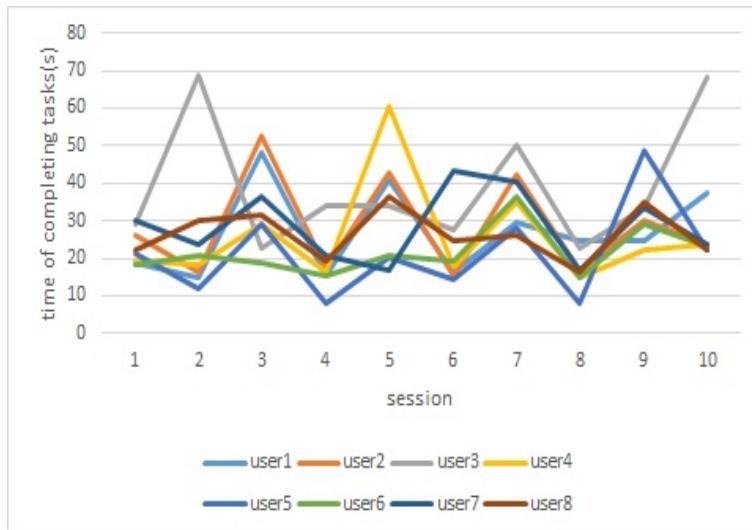


図 26: タッチ操作で行った場合のタスク完了時間

user	1	2	3	4	5	6	7	8	9	10
user 1	18.594	14.76	48.0895	15.471	40.895	16.112	29.361	24.825	24.568	37.239
user 2	26.341	16.304	52.806	17.455	42.814	15.387	42.525	16.75	30.207	23.626
user 3	29.064	68.713	22.504	33.809	33.809	27.666	50.134	22.696	33.6	68.144
user 4	19.322	18.355	28.85	16.118	60.248	17.979	34.95	14.84	22.141	23.7
user 5	21.124	11.808	28.853	8.113	20.159	14.152	28.341	8.213	48.636	22.272
user 6	18.24	20.897	18.962	15.407	20.586	19.283	36.651	14.693	28.912	22.976
user 7	30.199	23.566	36.409	20.787	16.745	43.175	40.157	16.58	33.42	23.878
user 8	22.14	30.246	31.462	19.185	36.515	24.738	26.136	16.408	34.843	22.193

表 4: ユーザーごとのタスク完了時間 (タッチ操作の場合)

user	平均	平均 (6~10)	平均 (9~10)
user 1	26.99145	26.421	30.9035
user 2	28.4215	25.699	26.9165
user 3	39.0139	40.448	50.872
user 4	25.6503	22.722	22.9205
user 5	21.1671	24.3228	35.454
user 6	21.6607	24.503	25.944
user 7	28.4916	31.442	28.649
user 8	26.3866	24.8636	28.518

表 5: ユーザーごとのタスク完了時間の平均(タッチ操作の場合)

本システムの場合、最初のほうはタスク完了時間の値が大きい人が多かったが、最後のほうは操作に慣れてきて全員、50秒を切っていることが表2からもわかった。表3と表5より全体の平均はタッチ操作のほうが早いが、後半の平均を見るとそれぞれの差はかなり縮まっており、最後のほうは差がほとんどなくなり、人によっては本システムのほうが早いという結果になった。本システムのほうで中盤で時間が極端にかかった人がいた。練習しても本システムになかなか慣れなかったことや、途中で逆に慣れてきて操作がいい加減になっていることなどが原因なのではないかと考えられる。また、時々目の前に正解の駅があったとしても見落としてしまうことも考えられる。図25と図26からタッチ操作よりも本システムのほうがセッション数を増やせば増やすほどタスク完了時間が低下する傾向の人が多いことがわかった。

### 6.3 アンケート結果

ここではアンケートの結果を示す。

#### 6.3.1 スクロールの精度

なかなか本システムの操作に慣れずスクロール精度については悪いと思っている人が多かった。スクロールの精度の向上はもちろんだが、操作方法の説明、注意すべき点についてもう少し詳しく説明すべきだった。

1. とても良い	2. 良い	3. 普通	4. 悪い	5. とても悪い
0 人	1 人	1 人	5 人	1 人

表 6: スクロールの精度はどうでしたか？

#### 6.3.2 ズームの精度

スクロールに比べるとズームの精度はまだ良いと思っている人が多かったようだった。良いと答えた1人はズームに関してはタッチ操作と違って指が邪魔にならなくて良いと感じたようだった。

1. とても良い	2. 良い	3. 普通	4. 悪い	5. とても悪い
0人	1人	3人	4人	0人

表 7: ズームの精度はどうでしたか？

### 6.3.3 疲労感の有無

疲労感に関しては個人差が見られた。本システムに慣れた人は疲労感をほとんど感じなかったようだが、なかなか慣れなくて操作がうまくいかなかった人は疲労感を少し感じたようだった。

1. 全くない	2. あまりない	3. 普通	4. 少しあつた	5. かなりあつた
2人	0人	2人	4人	0人

表 8: 疲労感はありましたか？

### 6.3.4 スクロールやズームをしている際の画面全体の把握のしやすさ (タッチの場合)

全員普段からスマートフォンを使用していてタッチ操作に慣れているので特に気にならない人も多かったように見えた。

1. とても把握しやすい	2. 把握しやすい	3. 普通	4. 把握しづらい	5. とても把握しづらい
1人	1人	5人	1人	0人

表 9: タッチ操作の場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？

### 6.3.5 スクロールやズームをしている際の画面全体の把握のしやすさ (FingCV の場合)

本研究の目的である画面領域を最大限生かすという点はこの設問より改善されていると思った人が多かったことがわかった。しかし、3人はそう思わなかつたことがわかった。理由はカメラのプレビュー画面に集中しすぎてしまうことが原因で画面全体が逆に見えなくなってしまうからではないかと考えられる。

1. とても把握しやすい	2. 把握しやすい	3. 普通	4. 把握しづらい	5. とても把握しづらい
2人	3人	0人	3人	0人

表 10: 本システムの場合、スクロールやズームをしている際に画面全体を把握しやすかったですか？

### 6.3.6 タッチ操作と FingCV の比較

全員タッチ操作に慣れていたので本システムに関して違和感を感じた人が多かった。もう少しスクロールやズームの精度を向上できれば評価は違っただろう。

1. とても良い	2. 良い	3. 同じくらい	4. 悪い	5. とても悪い
0人	0人	0人	8人	0人

表 11: タッチ操作と比べて本システムはどうでしたか？

### 6.3.7 総合的な FingCV の評価

違和感を感じた人がほとんどだったが 1 人は本システムを良いと感じたようである。ゲームや地図アプリを使用している際に指が邪魔になるので本システムは使えるのではないかと思ったようである。

1. とても良い	2. 良い	3. 普通	4. 悪い	5. とても悪い
0人	1人	4人	3人	0人

表 12: 総合的に見て本システムはどうでしたか？

### 6.3.8 追加してほしい機能

- タップ (6 人)
- 進む、戻る (2 人)
- スクロールとズームに切り替えるのを無くしてほしい
- スクロール時とズーム時の切り替え (画面のタッチ) 位置を工夫してほしい
- ズーム位置を指定できるようにしてほしい
- タブ一覧の表示

### 6.3.9 その他

- カメラのプレビュー画面に集中しすぎてしまう (3 人)
- スクロールが逆のほうが良かった (2 人)
- モード切替時のフィードバックが欲しい (2 人)
- スクロール幅を指の振り幅と合わせて欲しい。ズームも同様 (2 人)
- スクロールをするときに手の形や位置に気を付けなければならないのが少し面倒に感じた
- スクロールがちゃんと認識されない時があった
- ズームは使いやすい。できれば上下ではなく左右がいい
- 端を長押ししている間画面が動くのは個人的に好き
- カメラが横についていると手が楽な気がした

- スマホのスクロール方法と似ていたので直感的に操作できた
- 本システムではタッチ操作と比べて指で隠れずに画面が見れた
- ゲームや地図アプリで使ってみたい
- 左手の長押しする場所が途中でわからなくなったり
- もっと他にもズームやスクロールの操作を行える実験をやってみたい
- 全体的なレスポンス(地図の表示)が悪いのでそれを直さないと語りづらい
- もっとスペックの高い端末使おう
- 精度をもっと上げてほしい
- 位置センサとの併用を考えてほしい
- どこを長押しすればいいのか明確にしてほしい
- 指を反対向きにするとタッチ操作と同じになる
- タッチ操作も最初は慣れるまで時間がかかったので、この操作も慣れることで使いやすさが変わっていくと思う
- 指先が円の中(デフォルト位置)に戻ったことを知らせてほしい
- カメラと指を近づけないと操作しにくいということに気づくまではまともに操作出来なかつた。そこに気づくと操作しやすい

#### 6.4 おわりに

本章では実験結果、アンケート結果について述べ、これらの結果についての考察を述べた。次章は本論文のまとめを述べ、今回の研究の改善すべき点や今後の展望について詳しく述べる。

## 7 おわりに

### 7.1 まとめ

最初の章では研究背景としてスマートフォンの画面の表示領域を最大限に生かすような研究が行われてきたことを述べ、それらの問題点を解決するために本研究では端末の内蔵カメラを用いて画面の表示領域を狭める事なく、スマートフォンを操作する手法を提案、実装、評価することを本研究の目的として述べた。

第2章では関連研究として背面から端末を操作する研究と内臓カメラを用いて操作する研究の特徴や問題点について述べた。

第3章では本システムである FingCV の概要として仕様や操作方法を説明した。その後、スワイプ操作、ピンチイン、ピンチアウト操作、閲覧履歴の移動操作の3つの操作モードについて詳しく説明をした。

第4章ではまず FingCV の実装環境について述べ、その後に指認識について肌色領域の抽出の方法、手の検出の方法、指先の検出の方法について詳しく述べた。その次にブラウザの操作処理について、操作モードの切替方法、指先のジェスチャ検出の方法について詳しく説明をした。

第5章では FingCV についての評価実験の説明をした。まず、実験目的や被験者情報について述べた後、実際に行った実験内容について説明をし、最後にアンケートの内容について述べた。

第6章では実験の結果と考察について述べた。最初に、練習のべき乗則について詳しく説明をし、実験の測定結果を FingCV の方法とタッチ操作の方法についてグラフで表示した。FingCV のほうがセッション数を増やすほどタスク完了時間が低下することが実験結果からわかった。その後にアンケートの結果について述べた。なかなか操作に慣れない人も多く、スクロールやズームの精度がそこまで良くなかったため、FingCV の総合的な評価としてはあまり良くなかった。個人差もかなりあったように見られた。画面の表示領域を最大限生かすという点は改善されたという意見が多くなったことがわかった。しかし、カメラのプレビュー画面に集中しすぎて画面全体が見づらくなってしまうという問題点も挙がった。

### 7.2 改善すべき点

実験結果よりスクロールやズームの精度があまり良くなかったことがわかったため、指認識についてもう少し改める必要があると感じた。画像処理を使って実装しているため背景の色や照明に気を付けなければならないこともあったので、誤認識を減らすような処理がもっと必要であると痛感した。

ブラウザの操作処理についてもスクロールやズームの幅をもう少し自由に設定できるようにすべきだったと感じた。操作モードの切替時のフィードバックについてもあったほうがわかりやすいと思った。また、スクロールの方向をタッチ操作と同様にしたが、逆のほうが良いという意見もあったので両方の方法でできるように実装すべきだった。

### 7.3 展望

改善すべき点も多かったが、内蔵カメラだけでスマートフォンの操作をするのは困難な場面も多く、限界を感じた。アンケートの結果でもあったが、位置センサ等の独自のハードウェアを付けたほうがより安定した処理ができる、もっと快適に操作ができるようになり、タッチ操作の方法と同等以上のシステムが作成できるのではないかと考えた。また、スマートフォンの処理能力は年々上がっているので、より新しい端末を使うことでもっと安定した操作が期待できるだろう。

## 謝辞

ご指導を頂いた赤池先生、角田先生並びに、実験の被験者やその他の面でも日々ご助力を頂いた研究室の皆様方に厚くお礼を申し上げます。

## 参考文献

- [1] Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Brnwell, Chia Shen: LucidTouch:A See-through Mobile Device, UIST' 07 Proceedings of the 20th annual ACM on symposium on User interface software and technology, p.269-278 (2007)
- [2] 小林 茂, 鈴木 宣也, 赤羽 亨, 蝶田 直, 近藤 崇司, 伊豆 裕一, 米山 貴久, 横内 恭人: 裏タッチインターフェース 画面の背面から操作するインターフェースの提案, インタラクション論文集 2010, SA24 (2010)
- [3] 竹田 智, 岩田 満: 内蔵カメラを用いたスマートフォン操作手法の提案, 平成 24 年度電子情報通信学会東京支部学生会研究発表会論文集, p.205 (2013)
- [4] 竹田 智, 岩田 満: 内蔵カメラを用いたジェスチャによるスマートフォン操作手法の検討, 情報処理学会第 76 回全国大会論文集, 4-25,26 (2014)
- [5] 岡田浩臣, 星野孝総: HMD を用いた仮想ガジェットの開発, ViEW ビジョン技術の実利用ワークショップ講演論文集 2012, ROMBUNNO.IS2-D3(2012)
- [6] Subway Map Visualization jQuery Plugin, <<http://kalyani.com/2010/10/subway-map-visualization-jquery-plugin/>>
- [7] canvas を使って地下鉄の路線図を描ける jQuery プラグインで東京の地下鉄の路線図を描いてみた, <<http://webdrawer.net/javascript/subwaycanvas.html>>

## 付録 本実験アンケート

本実験後に回答してもらった各被験者のアンケート用紙を添付する。