修士学位論文

論文題目　脊髄損傷患者の為の

ヒューマンインタフェースの開発

ふりがな
氏　　名　たにしま　あやの　谷島　綾乃

専　　攻　機械工学専攻

指導教授　金丸　隆志　准教授

修了年月（西暦）　2015年3月

工学院大学大学院

論文要旨

1. 緒言

日本には脊髄損傷患者が 10 万人以上おり、年間約 5 千人が新たに脊髄損傷を負っている。患者の年齢別分布では、交通事故やスポーツ、労働災害を原因とする 20 代と、転倒などを原因とする 50 代～60 代に二極化したピークがあり、若年及び高齢重度脊髄損傷患者の増加が問題となっている。脊髄損傷は体に麻痺が残りやすい病態であり、脊髄の損傷箇所に伴い麻痺する肢体部位が変化するため、一度傷つくと二度と再生することができない。

また重度の病態を抱える脊髄損傷患者は、身体の自由が極端に制限される。そのため、リハビリテーション病院での長期療養を余儀なくされ、基本的な生活作業にも介助を必要とする。近年、患者の QOL（Quality of Life） の向上が重要だと考えられており、患者がより豊かな生活をおくるために多くの福祉機器が提案されている。

現在、脊髄損傷患者がパソコンの操作を行う際にはマウススティックが広く使用されている。マウススティックの片方を口にくわえ、もう片方をパソコンの画面やキーボードに押し付けることで操作を行う。そのため、長時間の使用は患者の歯やあごに大きな負担がかかり、その状態が続くと開咬と呼ばれる歯の変形を引き起こす。また、何らかの要因によってマウススティックが口から離れてしまった場合は、自力での対応が困難であり、介護者による再調整が必要となる。そこで器具への接触を行わずに、操作を行うことが可能なシステムが必要であると考えた。

本研究では、損傷レベル C4 の患者を対象としたタブレット用のヒューマンインタフェースを開発する。患者に負担をかけずに操作できるようなヒューマンインタフェースを開発することによって、操作による病気の防止と QOL の向上を目的とする。具体的には、息と顔の動きを用いて、タブレットのタッチレス化を行う。以下でこのシステムの機能の解説を行う。

2. 仕様と構成

本研究で開発するヒューマンインタフェースは、病院に入院中や在宅医療中などに使用されることを想定している。しかし脊髄損傷患者に対して、いつも看護師や家族の目が届いている状況とは限らない。その様な状況では、マウススティックのマウスピース部を飲み込むなどの理由で、万が一喉を詰まらせた場合は致命傷になってしまう可能性が高い。患者に負担をかけずに操作を行うためにも、そして衛生を保つためにも器具への接触を行わずに操作を行うことが可能なシステムが必要であると考えた。以上により、本研究で開発するシステムは下記の条件を満たす必要がある。

【条件 1】脊髄損傷患者による操作を検知できる
【条件 2】安全である
【条件 3】衛生が良い（＝清潔を保てる）
【条件 4】直感的に操作ができる
【条件 5】コストが安い（＝病院などに提供できる値段である）
【条件 6】患者のプライバシーが守られる（＝操作内容を外部へ漏らさない）

全ての条件を満たすには 1 つのセンサでは補いきれないと考えられる。そのため、本研究では

息と顔の動きを組み合わせて、タブレットのタッチレス操作を実現する。

本研究で提案するヒューマンインタフェースは、大きく分けて 3 つの構成からなる。まず、圧力センサで息の圧力（息圧）を検知し、タブレットのスリープを解除する。次に、タブレットのカメラから映像を取得し、画像処理の手法であるオプティカルフローを用いて顔の移動方向と距離を計算する。また、Haar-like 特徴分類法により顔の位置を検出する。そして、顔の位置と移動に応じてマウスカーソルの移動を行い、顔の位置を利用してマウスクリックを行う。

3. 息によるタブレットのスリープ解除

息の圧力（息圧）を計測するための圧力センサ（息圧センサ）に接続した内径 30mm の太いチューブに、50mm 離した位置から息を吹きかける。その時の息圧センサの出力電圧を Arduino で読み取り、オシロスコープで確認する。20 代の成人男性を対象に実験を行ったところ、個人差があり、1.21V～1.29V（40Pa～120Pa）であることがわかった。

実験で得られたデータを元に閾値を決め、一定時間内に規定の回数閾値を超えた場合に Arduino からタブレットに信号を送り、タブレットのスリープを解除する。想定するインターフェースの使用者には高齢者も含まれるため、息圧も小さくなることが予想される。そこで最も小さかった人のデータを元に閾値を 1.21V（40Pa）、閾値を数える時間窓を 1500ms、その時間内に閾値を超えるべき回数を 3 回と定めた。また、センサで取得した値はアナログ値であり、A/D 変換後のデジタルの値に対する閾値を定めると 248 となる。

プロトタイプとして Android タブレットを用いてスリープの解除を行った。Android タブレット上では、Arduino と USB を通して ADK という仕組みで通信を行う。そして、Arduino から信号が送られてきた場合に Android タブレットをスリープ状態から復帰させ、スクリーンロックを解除する。解除後アプリを起動させて、画面上に Status ON とテキストを表示する。

プロトタイプである Android タブレットのスリープ解除を行うことができたため、次に Windows タブレットにおけるスリープ解除を行った。Windows タブレットにおけるスリープ解除では、キーボードやマウスエミュレータ搭載の Arduino Leonardo からの信号により、スクリーンセーバーを解除する。

4. 息圧センサによるマウカーソルの操作

タクトスイッチによるプロトタイプを作成後、患者が一定方向に首を動かしづらい場合に、その向きのカーソル操作を息圧センサで代用するために、息圧センサによるマウカーソルの移動プログラムと回路を作成した。息圧を用いて、Windows タブレットにおけるマウスクリックと上下左右の 4 方向へカーソルの移動を行う。クリックについては、タブレットのスリープ解除の際に達成しているため、本章では各センサが 4 方向のカーソル移動へそれぞれ対応できるようにした。

5. 機構

息圧センサを接続したチューブを患者の口元付近で固定するための器具を作成する。器具は大きく 3 つの部位に分かれており、ベッドに固定する土台である「固定部」、固定部から患者の口元への距離を調節する「調節部」、患者が息を吹きかける「吹きかけ口」である。固定部で机やベッドの柵を挟み込むようにして固定を行い、その上に回転機構が乗る。さらに、チューブ内に太さ 4mm の針金を通すことでフレキシブルに調整が行うことのできる調節部が伸び、その先端に吹きかけ口が設置される。各部位の自由度は、固定部が 1 自由度、調節部が 6 自由度、吹きかけ口が 1 自由度である。本体総重量は 825g であり、軽量である。

6. 顔の移動によるマウスカーソルの操作

タブレット上でマウスカーソルの移動を行うために、タブレットのカメラから映像を取得し、特徴点を抽出する。特徴点はエッジ抽出を行い、画像中の線の端点（線の終点)、交差点すなわち画像の境目や角、色彩が大きく変わる濃淡の濃い部分や輪郭などが用いられる 。その後、画像処理の手法の 1 つであるオプティカルフローを用いて顔の移動方向と距離を計算する。患者の移動に応じて上下左右へマウスカーソルの移動を行う。本研究の対象である患者はベッドで多くの時間を過ごすため、患者の背後の物や人が移動することは、殆どない。よって本研究では画面内の動きは患者の動きであると考えることができる。

WindowsPC 上で動作確認を行ったところ、オプティカルフローの平均ベクトルの向きに従い、マウスカーソルを移動することができた。しかし首の可動域などの理由から画面の端へのカーソル移動が難しい問題と、移動や停止範囲を使用者が判断しにくい理由からマウスカーソルの停止や移動方向の切り替えが難しい問題が浮かび上がった。そこで移動ベクトルに顔の位置情報を組み合わせ、その情報を視覚化することで問題を解決する。

7. 顔の位置によるマウスカーソルの操作

Haar-like 特徴分類法を用いて顔と口の検出を行い、その情報を元にマウスカーソルの移動を行うシステムを開発した。Haar-like 特徴では顔と口それぞれ検出できるので顔の位置によるマウス移動と口の位置によるマウス移動の両方を検討し、最終的には安定性の高い顔の検出を用いることにした。ただし、口は顔の他の部位と色が異なるため、ユーザーが認識しやすく、顔の中の総体的な位置の個体差が少ないことから、口の位置にカーソル移動の目安があるとわかりやすいと考えた。

ウィンドウを 8 つの領域に分け、カーソル移動の目安となる赤色の丸がある領域に応じてマウス操作を行う。通常のカーソル移動では、細かな左右のカーソル移動を行う。例えば、赤色の丸の位置が右カーソルの領域内にあり、かつオプティカルフローで検出した物体が右に動いている間、カーソルを右へ少し動かす。カーソル移動の連続動作では、赤色の丸が領域内にある間、その領域にて指定されている方向へカーソルを移動させ続ける。また、クリックの領域内に赤色の丸が一定時間入った場合にクリックを行う。色覚の多様性に配慮し、カラーユニバーサルデザインを用いた操作盤を作成した。

8. 結言

脊髄損傷のような身体の自由が極端に制限される患者の QOL を向上するために、息と顔の動きを用いてタブレットを操作するヒューマンインタフェースを作製し、タブレットのタッチレス化を行った。

目次

第1章　緒言

1.1 研究の背景及び目的

日本には脊髄損傷患者が 10 万人以上おり、年間約 5 千人が新たに脊髄損傷を負っている[2]。患者の年齢別分布では、交通事故やスポーツ、労働災害を原因とする 20 代と、転倒などを原因とする 50 代～60 代に二極化したピークがあり、若年及び高齢重度脊髄損傷患者の増加が問題となっている （図 1.1）[1～3]。脊髄損傷は体に麻痺が残りやすい病態であり、脊髄の損傷箇所に伴い麻痺する肢体部位が変化するため、一度傷つくと二度と再生することができない[2]。

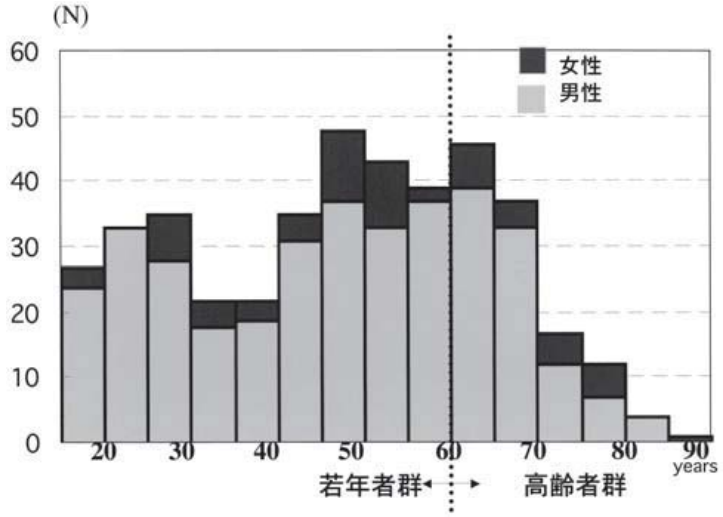


図 1.1: 脊髄損傷の受傷時年齢[1]

また重度の病態を抱える脊髄損傷患者は、身体の自由が極端に制限される。そのため、リハビリテーション病院での長期療養を余儀なくされ、基本的な生活作業にも介助を必要とする。近年、患者の QOL（Quality of Life） の向上が重要だと考えられており、患者がより豊かな生活をおくるために多くの福祉機器が提案されている。例えば、マウススティック（徳器技研工業株式会社）[4]、伝の心（株式会社日立ケーイーシステムズ）[5]、マイトビーC15Eye（Tobii Technology）[6] 、あご操作マウス（吉備高原医療リハビリテーションセンター）[7] などのようなパソコン操作用のインターフェースが挙げられる（図 1.2～1.5）。患者のための福祉機器にパソコンを操作するインターフェースが多い理由として、文章や写真の閲覧、音楽、メールなど多目的に使用でき、患者の QOL 向上にパソコンが貢献しやすいためであると考える。

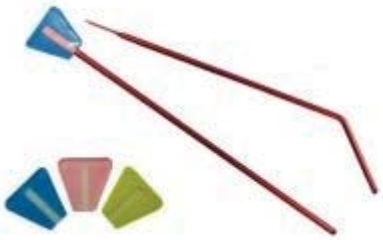


図 1.2: マウススティック[4]

図 1.3: 伝の心[5]

図 1.4: マイトビーC15Eye[6]

図 1.5: あご操作マウス[7]

現在、脊髄損傷患者がパソコンの操作を行う際には図 1.2 のマウススティックが広く使用されている。これはマウススティックが安価かつ、仕組みが単純なためである。図 1.6 のようにマウススティックの片方を口にくわえ、もう片方をパソコンの画面やキーボードに押し付けることで操作を行う。そのため長時間の使用は患者の歯やあごに大きな負担がかかり、その状態が続くと開咬と呼ばれる歯の変形を引き起こす（図 1.7）[45]。また、何らかの要因によってマウススティックが口から離れてしまった場合は、自力での対応が困難であり、介護者による再調整が必要となる。そこで器具への接触を行わずに、操作を行うことが可能なシステムが必要であると考えた。

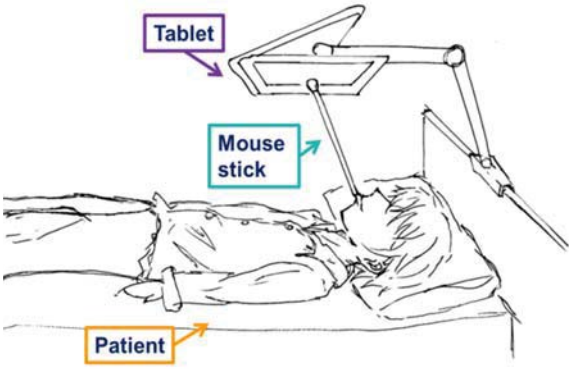


図 1.6: タブレットを操作する様子

図 1.7: マウススティックによる開咬[45]

本研究では、損傷レベル C4 を対象としたタブレット用ヒューマンインタフェースを開発する。また、患者に負担をかけずに操作できるようなヒューマンインタフェースを開発することによって、操作による病気の防止と QOL を向上することを目的とする。

操作の対象をタブレットとパソコンとしたのは、患者の多様な嗜好に対応できる汎用性の広い点と、持ち運びが容易で外出先でも使用ができる点からである。また、タブレットの基本操作は以下の 3 動作からなる。

【動作 1】スリープ解除
【動作 2】マウスカーソルの移動
【動作 3】タッチ（選択）

動作 3 は、タッチを画面等への物理的接触で実現することは好ましくない。そのため、図 1.8 のように患者の息の圧力（息圧）と顔の動きを用いてタブレットのタッチレス化を行う。

図 1.8: システム構成

1.2 本論文の構成

本論文の構成は以下の通りである。第 2 章「脊髄損傷」では脊髄損傷のメカニズムについてまとめる。第 3 章「仕様」では本研究で開発するタブレット用インターフェースの仕様を解説する。第 4 章「タブレットのスリープ解除」では息圧センサを用いた Android タブレットと Windows タブレットのスリープ解除についてまとめる。第 5 章「機構」ではヒューマンインタフェースの機構と配置について記す。第 6 章「息圧によるマウスカーソルの操作」では息圧を検出するセンサ（息圧センサ）によるマウスカーソル操作についてまとめる。第 7 章「画像処理によるタブレットのカーソル移動」では画像処理を用いたカーソルの移動とマウスクリックについて述べる。第 8 章「評価」ではでは学内におけるヒューマンインタフェースの評価をまとめる。第 9 章「結言」では今回の研究で成し得たことをまとめる。

第 2 章　脊髄損傷

2.1 脊椎と脊髄

2.1.1 脊椎

脊椎（脊柱）は、椎骨と呼ばれる骨が連なってできており、頚椎（7 個)、胸椎（12 個)、腰椎（5 個)、仙椎（5 個、ただし形状は仙骨として 1 個）の 4 つの領域に区分けされている。脊椎の模式図を図 2.1 に示す。脊椎の中心を通る脊柱管の中に脊髄があり、椎骨によって保護されている。椎骨と椎骨の間には椎間板（軟骨）があり、脊椎への衝撃を和らげている[8,9] 。

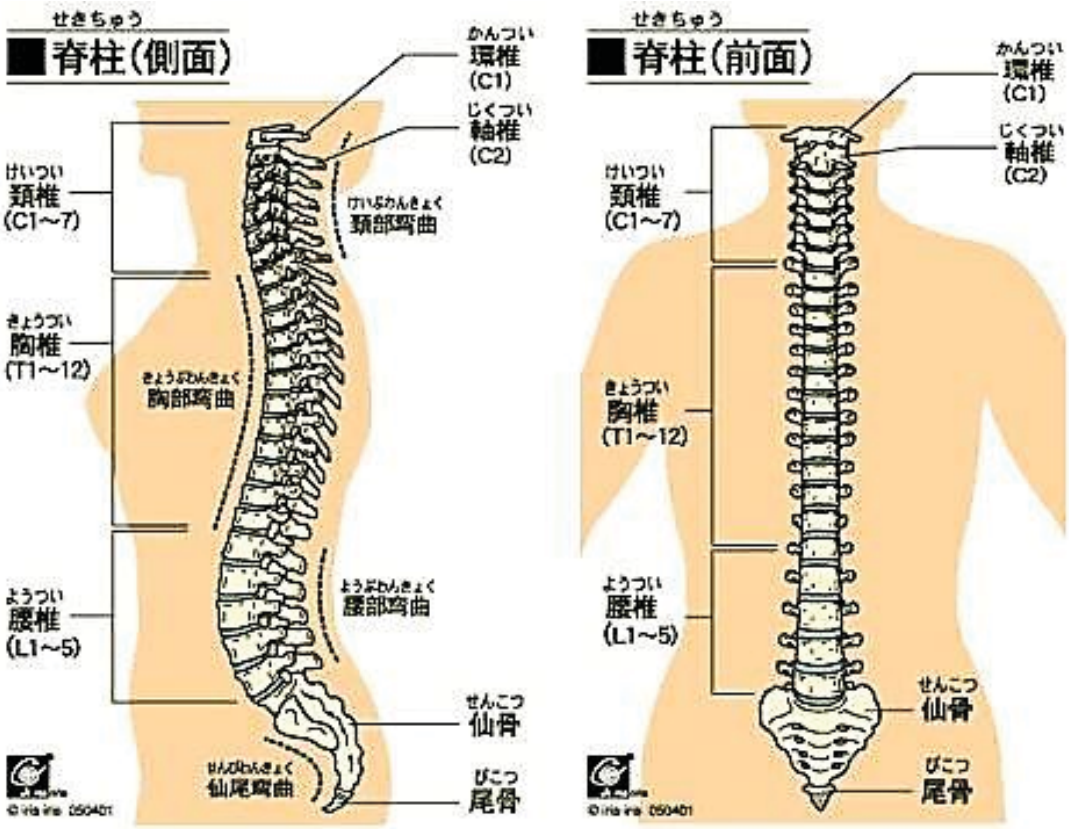


図 2.1: 脊柱の全景とその区分[11]

2.1.2 脊髄

脊髄は長く、傷つきやすい管状の構造物である。脊髄は内側から軟膜、くも膜、硬膜と呼ばれる 3 層の髄膜によって保護されており、軟膜は脊髄に密着し、軟膜とくも膜の間は脳脊髄液で満たされているため、脊髄は脳脊髄液の中に浮いた状態となる。よって、通常の状態では脊髄は圧迫を受けることはない。脊髄の髄膜を図 2.2 に示す。脊髄も脊椎と同じく、それぞれ頸髄、胸髄、腰髄、仙髄の 4 つの領域に区分けされている。脊髄からは、椎骨と椎骨の間を通り、31 対の脊髄神経[頸神経（8 対)、胸神経（12 対)、腰神経（5 対)、仙骨神経（5 対)、尾骨神経（1 対）] がそれぞれ、神経根に分かれる。脊髄の前側にあるのが運動神経根であり、脳と脊髄からの命令を、体の他の部分、特に骨格筋へ伝える。一方、脊髄の後ろ側にある、感覚神経根は体の他の部分の情報を脳へ伝える。また脊髄は、脊椎の下方約 4 分の 3 の位置で終わるが、そこから下へは神経の束がひと束伸びており、この神経の束（馬尾）は、下肢の運動・感覚を伝える。脊髄の構造を図 2.3 に示す[8]。

図 2.2: 脊髄の髄膜[8]

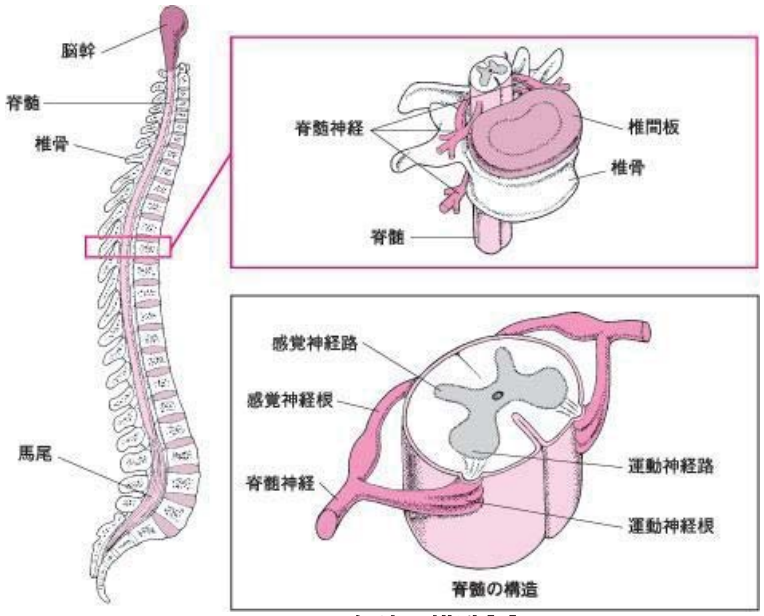


図 2.3: 脊髄の構造[8]

2.2 脊髄損傷

2.2.1 一般的症状

脊髄損傷とは脊椎に対して、交通事故、高所よりの落下、落下物、スポーツなどによる強い外力が加わることで、背骨にある脊柱管という骨の管の中を通る、神経の束である脊髄が損壊し、損傷を受けることである。その損傷状態により、完全型と不完全型に分離される。完全型（完全麻痺）は、脊髄が横断的に離断した状態で、神経伝達機能が完全に断たれる為、その部位以下を動かすことは出来ず、感覚はない。一方、不完全型（不完全麻痺）は、脊髄の一部が損傷し、機能が一部残存する。以下、特に注釈がない場合は、完全型についてのみ記述する。

なお、脊髄には番号が割り振られており、それぞれ支配するデルマトーム（皮膚知覚帯）が異なる（図 2.4)。脊髄を損傷すると対応するデルマトーム以下の感覚及び、損傷部位以下の運動機能（表 2.1、2.2 参照）を失い、麻痺状態となる。しかし、実際には本来感じないはずの痺れなどの異常知覚、肢体切断時と同様に幻肢痛を感じることがある。

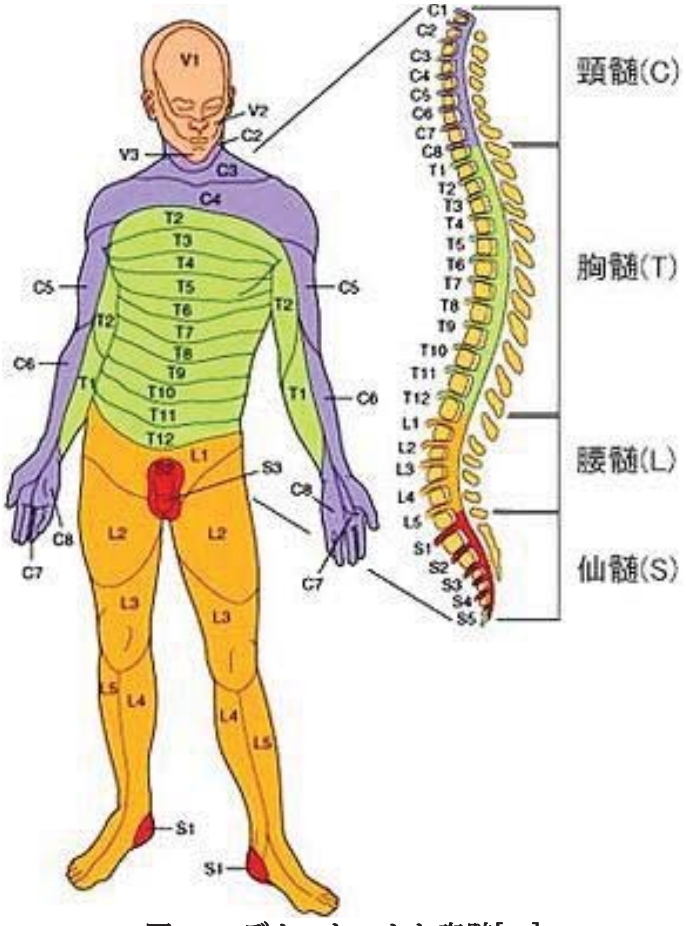
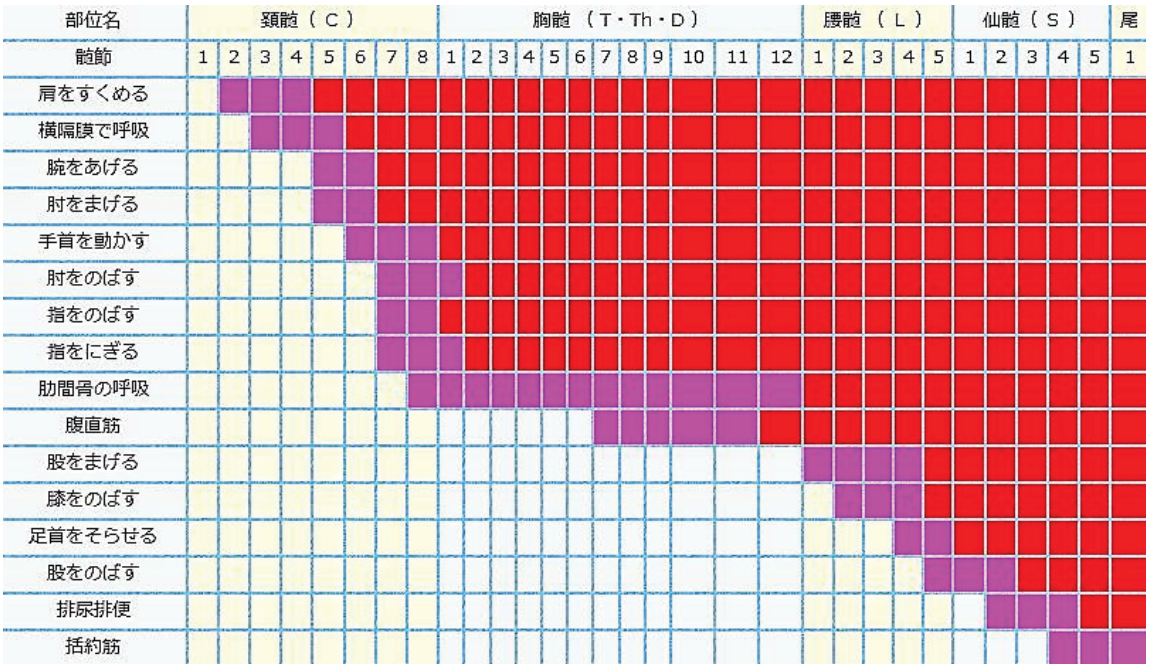


図 2.4: デルマトームと脊髄[11]

表 2.1: 脊髄損傷と運動機能 1[12]

損傷箇所	領域	支配筋	残存機能
頸椎	C1～C2	高位頸筋群	首の運動
	C3～C4	胸鎖乳突筋 僧帽筋 横隔膜	首の運動 肩挙上、上肢屈曲、 外転（水平以上） 吸息
	C5	肩甲骨筋群 三角筋 上腕二頭筋 腕橈骨筋	上腕屈曲外転 肩関節外転 肘関節屈曲 肘関節屈曲
	C6	橈側手根屈筋 円回内筋	手関節背屈 手回内
	C7	上腕三頭筋 橈側手根屈筋 総指伸筋	肘関節伸展 手関節屈曲（掌屈） 手指伸展
頸椎 胸椎	C8～T1	手指屈筋群 手内筋群	こぶしをにぎる 母指対立保持、 つまみ動作、 手指外転内転
胸椎	T2～T7	上部肋間筋群 上部背筋群	強い吸息 姿勢保持
	T8～T12	下部肋間筋群 腹筋群 下部背筋群	強い吸息 有効な咳 座位姿勢保持
腰椎	L1～L3	腰方形筋 腸腰筋 股内転筋群	骨盤挙上 股関節屈曲 股関節内転
	L3～L4	大腿四頭筋	股関節伸展
腰椎 仙椎	L4、L5、S1	中殿筋 大腿二頭筋 前脛骨筋	股関節外転 膝関節屈曲 足関節背屈（踵歩き）
	L5、S1～S4	大殿筋 腓腹筋	股関節伸展 足関節底屈 （つま先歩き）
仙椎	S1～S4	肛門括約筋	排便、排尿コントロール

表 2.2: 脊髄損傷の運動機能 2[13]



■で表した部分は、動くか動かないか、はっきりしない部位
■で表した部分は完全に動かす事が可能である部位

2.2.2 受傷原因と比率

1990～1992 年の調査に基づき、受傷原因と平均年齢を表 2.3、交通事故の種別を表 2.4、年齢帯別受傷者数を表 2.5、受傷時の年齢を図 2.5、損傷部位別受傷者数を表 2.6 に示す。表 2.3 に示されているように、受傷平均年齢は 48.6 歳であり、その内訳は交通事故やスポーツ、を原因とする 20 歳代と転倒などを原因とする 50 代～60 代に二極化したピークがある。また、怪我の部位は、より深刻な頸髄損傷がそのうち 75 %を占める[2]。

表 2.3: 受傷原因と平均年齢（1990～1992 年）[2]

原　因	1990 年	1991 年	1992 年	合計(例)	比率(%)	平均年齢(歳)
交通事故	1547	1308	1408	4263	43.7	44.4
高所転落	10　2	912	894	2818	28.9	53.2
転　倒	407	417	436	126	12.9	61.7
打撲･下敷き	209	169	159	537	.5	48.3
スポーツ	182	174	172	528	5.4	28.5
自殺企図	60	63	44	167	1.7	31.9
その他	48	54	7	179	1.9	47　3
合　計	3465	3097	3190	9752	100	48　6

表 2.4: 交通事故の種別（1990～1992 年）[2]

種　別	受傷者数(例)	比率(%)
自動車	2007	47.1
オートバイ	1239	29.1
自転車	66	15.6
歩行者	250	5.9
その他	104	2.4
合　計	4263	100

表 2.5: 年齢帯別受傷者数（1990～1992 年）[2]

年齢帯	頚　損	胸･腰･仙	不　明	合計(例)
0－15	119	40	2	161
16－30	1167	878	10	2055
31－45	117	542	3	1716
46－60	2250	597	8	2855
61－75	2112	71	9	2391
76－96	492	63	3	559
合　計	7311	2391	5	9737

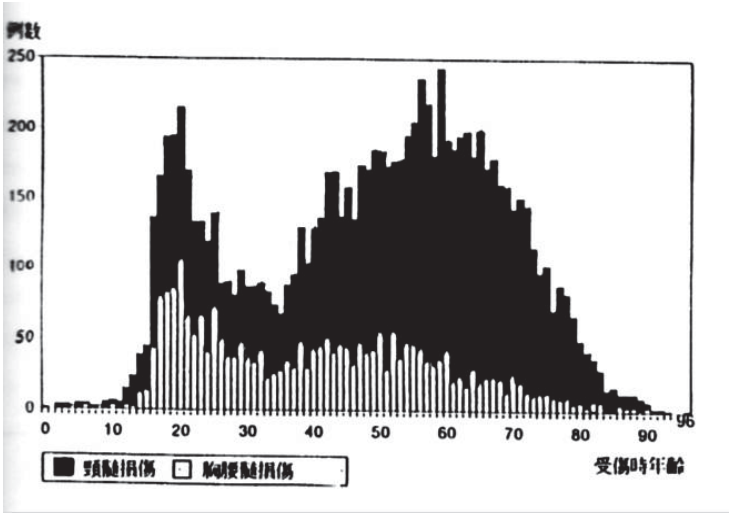


図 2.5: 受傷時の年齢[2]

表 2.6: 損傷部位別受傷者数（1990～1992 年）[2]

損傷位置	受傷者数(例)	比率(%)
頸髄損傷	7317	75
胸腰仙髄損傷	2408	24.7
不　明	27	0.3
合　計	9752	100

2.2.3 リハビリテーション

中枢神経である脊髄は回復することがないため、脊髄損傷のリハビリテーションは一般的なものとは異なり、失われた機能を回復させるものではない。ここでのリハビリテーションとは、残存機能を用いていかに日常生活動作（ADL： Activities of Daily Living）を可能とするか、言い換えるならば、残存機能を脊髄を損傷する以前より活用することで、失った機能を補完し、日常動作を行えるようにすることである[14,15]（図 2.6）。

1991 年から 2001 年までに福岡県飯塚市の総合せき損センターに受傷後 7 日以内に搬送され、6 ヶ月以上経過観察された 430 例の、入院時と退院時を改良フランケル分類（表 2.7 参照）で比較したものを表 2.8 で示す[16]。

図 2.6: リハビリの様子[18,19]

表 2.7: 改良フランケル分類[16]

A. motor, Sensory complete 運動・知覚完全麻痺 仙髄の知覚脱出・肛門括約筋完全麻痺
B. Motor complete, sensory only 運動完全（ 下肢自動運動なし), 感覚不全 B 1 触覚残存（ 仙髄領域のみ） B 2 触覚残存（ 仙髄及び下肢にも残存する） B 3 痛覚残存（ 仙髄あるいは下肢）
C. Motor useless 運動不全で有用ではない（ 歩行不能） C 1 下肢筋力 1・2（ 仰臥位で膝立てが出来ない） C 2 下肢筋力 3 程度（ 仰臥位で膝立てが出来る）
D. Motor useful 運動不全で有用である（ 歩行出来る） D 0 急性期歩行テスト不能例 下肢筋力 4・5 あり歩行出来そうだが、急性期のため正確な判定困難 D 1 車椅子併用例 屋内の平地であれば 10m 以上歩ける（ 歩行器装具杖を利用して良い）が 屋外・階段は困難で日常的には車椅子を併用する ※ 10m 以下の歩行であれば〔C2〕と判定※ D 2 杖独歩例あるいは中心性損傷例 杖独歩例：杖・下肢装具など必要であるが屋外歩行も安定し車椅子不要 中心性損傷例：杖・下肢装具など不要で歩行は安定しているが 上肢機能が悪いため、入浴や衣服着脱などに部分介助を必要とする D 3 独歩自立例 筋力低下，感覚低下はあるが独歩で上肢機能も含めて日常生活に介助不要
E. Normal 正常 神経学的脱落所見なし（ 自覚的痺れ感・反射の亢進あり）

膀胱機能は包含せず（通常 D 以上では自排尿である)。左半身と右半身とで差のある場合には、左右各々を評価する（左 B2、右 C1 など)。判定に迷う時には悪い方に入れる。D0 群は実際 D1、D2、D3 のいずれかであるので、予想できれば D0（D1） や D0（D2） と記載する。

表 2.8: 麻痺予後の推移[16]

麻痺レベル 【最終時→】 【入院時↓】	A	B	C1	C2	D1	D2	D3	E
A	140	18	18	18	4	.	2	.
B1	1	3	1	1	3	.	.	.
B2	.	6	6	10	5	4	1	.
B3	.	.	1	2	7	5	.	.
C1	.	.	6	14	13	17	5	.
C2	.	.	.	2	18	16	11	.
D	.	.	.	.	1	17	46	8

第 3 章　仕様

3.1 現状におけるパソコン操作用の福祉機器と問題点

第 1 章で述べた通り、伝の心（株式会社日立ケーイーシステムズ）[5]、マイトビーC15Eye（Tobii Technology）[6] など、近年パソコンを操作するために多くの福祉機器が提案されている。しかし、高価であることや操作に慣れるまで練習が必要なことから、十分に普及が進んでいない。

一方で、マウススティックは低価格かつ直感的に操作が行え、かつ汎用性が高い。マウススティックとは、患者が日常用品や電気機器を口で操作するための自助具である。そのため多くの脊髄損傷患者に使用されており、パソコン操作以外の用途でも「患者の手となり、行動のほとんどを代行するもの」とまで言われている。しかしスティックの一方を銜え、もう一方で操作を行うため、対象物との距離に合わせて口で持ち替える必要がある（表 3.1）。スティックが口から離れてしまった場合は、予備のスティックを用いるか、介護者に頼まなくてはならない。その上、長時間の使用によって患者の歯やあごに大きな負担がかかり、顎の痛みや頭痛、そして開咬などの病気を引き起すことが問題になっている。また、マウスピース部を誤って飲んでしまうこともあり、健康面で不安をかかえる患者もいる[22]。

3.1.1 開咬

開咬（オープンバイト） とは、上下の歯を噛み合わせた時に、奥歯は噛んでも前歯が噛まない（前歯部開咬）、または、前歯は噛んでも奥歯が噛まない（臼歯部開咬） 歯並びのことである（図 3.1）[45]。開咬には見た目の問題や前歯でものを噛み切れないという問題以外にも、正しい発音がしにくかったり、奥歯に過剰な負担がかかってしまったりするという問題もある。開咬を治療するためには一般的に矯正を行うが、その際に抜歯をしなければならないケースも多々ある。開咬の原因にはマウススティックによるもの以外に遺伝的なものや、指しゃぶり、舌を前に出すなどのクセによるものなど様々なものがある[23]。

マウススティックによる開咬の場合は、図 3.2 のように長時間スティックを銜えることで矢印方向に力が加わり、歯が変形する。

表 3.1: 用途によるマウススティックの先端部の違い[22]

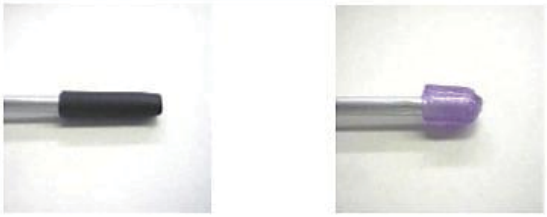
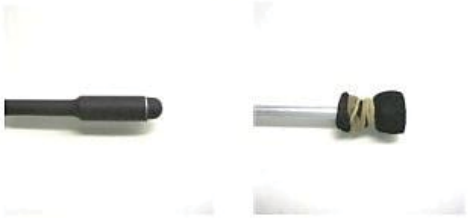
①本や新聞等のページ、紙面をめくりたい場合。 滑りにくい柔軟性のあるゴム素材。 （ゴムパイプ、『おゆまる』など）	 **【作り方】** ①ゴムパイプ： 先端に差し込む。 ②『おゆまる』： お湯で柔らかくして、先端で丸める。
②キーボード、リモコン等のボタンを押したい場合。 滑りにくい先端が細く硬い素材。 （木、オストロン、自由樹脂など）	 **【作り方】** ① 木 ： 穴を開けてパイプを通して接着、先端を尖らせる。 ②オストロン ： 2剤を調合し、先端で固めながら形を整える。 ③『自由樹脂』： お湯で柔らかくして、先端で丸める。
③スマートホン等のタッチパネルを操作したい場合。 柔軟性のある導電性のある素材。 （スタイラスペン、導電スポンジなど）	 **【作り方】** ①スタイラスペン： 使いやすい長さでカットして先端に接着する。 ②導電スポンジ ： 先端に輪ゴムやテープで巻きつける。

図 3.1: 開咬[45]

図 3.2: 開咬のメカニズム

3.2 仕様

3.2.1 ヒューマンインタフェースの仕様

本研究の対象は、脊髄損傷でも重度（フランケル分類 B，損傷レベル C4）の首から下が動かせない患者である（表 2.7）。患者の多様な嗜好に対応ができ、かつ汎用性の広がっていく可能性が高いタブレットを操作の対象とする。図 3.3 にタブレットの将来性の一例として、車の操作や走行が制御できる RoboCar PHV（ZMP、日本マイクロソフト）[24～26]を示す。

図 3.3: RoboCarPHV（[24,25]）

本研究で開発するヒューマンインタフェースは、病院に入院中や在宅医療中などに使用されることを想定している。しかし脊髄損傷患者に対して、いつも看護師や家族の目が届いている状況とは限らない。その様な状況では、マウススティックのマウスピース部を飲み込むなどの理由で、万が一喉を詰まらせた場合は致命傷になってしまう可能性が高い。患者に負担をかけずに操作を行うためにも、そして衛生を保つためにも器具への接触を行わずに操作を行うことが可能なシステムが必要であると考えた。以上により、本研究で開発するシステムは下記の条件を満たす必要がある。

【条件 1】脊髄損傷患者による操作を検知できる
【条件 2】安全である
【条件 3】衛生が良い（＝清潔を保てる）
【条件 4】直感的に操作ができる
【条件 5】コストが安い（＝病院などに提供できる値段である）
【条件 6】患者のプライバシーが守られる（＝操作内容を外部へ漏らさない）

現状のインターフェースが上記の条件にあてはまるか否かを考察してみよう。対象は、筋電位センサ、音声認識、アイトラッキングである。

●筋電位センサ

筋電位センサとは、筋肉を動かすときに脳から出る電気的信号を皮膚表面に固定した電極、すなわち表面電極で捕らえるセンサである。捕らえた信号を表面筋電位と呼ぶ。筋電位センサは、条件 1、2、4、6 を満たす。しかし、筋電位を計測するために皮膚に表面電極を貼る必要がある。そのため、電極を貼る人は筋肉の位置を把握しなくてはならず、また汗などで肌がかぶれる問題がある。よって条件 3 を満たさない。また、筋電位センサの価格は数十万円と高価であり、条件 5 も満たさない。

●音声認識

音声認識は条件 2～5 を満たす。また特徴的な言葉一つで検知可能なため一見、条件 1 を満たすように思えるが、使用する場所によってはテレビの音を拾ったり、または会話の中の単語を拾ったりして誤動作する可能性がある。また、操作の内容を声に出すために周囲の人にも操作内容が伝わってしまい、条件 6 を満たさない。

●アイトラッキング

アイトラッキングとは、赤外線装置やカメラなどから眼球の動きをトラッキングする手法である。この手法は患者と接触せずに眼の動きを計測するため、条件 1～3、6 を満たす。同様に、一見条件 4 の直感的な操作も行えように思えるが、視線を意識的に一点で固定することに慣れていない場合、操作をしている感覚がないため、意識的に操作をすることが難しい。また、無意識の視線移動にも反応してしまうという問題がある。条件 5 の価格について、以前はアイトラッキング機器は数百万円と高価であったものの、現在$99 の The Eye Tribe Tracker（The Eye Tribe）[22]など低価格の製品も販売されている（図 3.4）。

図 3.4: The Eye Tribe Tracker[21]

以上の結果を表 3.2 に示す。評価は 3 段階の○、△、×で記す。上記 3 手法に変わる手法として、本研究では息と顔の動きを組み合わせて、タブレットのタッチレス操作を実現する。全ての条件を満たしつつ、より直感的な操作を実現するには 1 つのセンサでは補いきれないと考えられるため、2 つのセンサを用いる。

表 3.2: 従来手法の評価

センサ	条件 1	条件 2	条件 3	条件 4	条件 5	条件 6
筋電位センサ	○	○	×	○	×	○
音声認識	△	○	○	○	○	×
アイトラッキング	○	○	○	△	△	○

3.2.2 ヒューマンインタフェースの構成

1.1 節で述べたように、本研究で提案するヒューマンインタフェースは、タブレットを操作するうえで最も基本的な動作である以下の 3 動作に特化して作成する。まず、圧力センサで息の圧力（息圧）を検知し、タブレットのスリープを解除する。次に、タブレットのカメラから映像を取得し、画像処理の手法であるオプティカルフローを用いて顔の移動方向と距離を計算する。また、Haar-like 特徴分類法により顔の位置を検出する。そして、顔の位置と移動に応じてマウスカーソルの移動を行い、顔の位置を利用してマウスクリックを行う。図 3.5 にシステム構成を示す。

図 3.5: システム構成

3.2.3 機構の仕様

本研究の対象である首から下を動かすことのできない C4 の患者が、ベッドに寝たままの状態で息によるスリープ解除を行うために、息圧センサに接続したチューブを患者の口元付近で固定するための機構を作成する。機構の作成における条件は下記の 2 つである。

【条件 1】チューブを患者の口元付近で固定できる
【条件 2】患者の 3 姿勢（正面、左、右）に対応できる（図 3.6）

条件 1 を満たすためには、患者の口元に合わせてチューブを任意の位置で固定できる必要がある。そのためには柔軟な調整を行うことのできる機構が望ましいと考える。また万が一、機構が壊れてしまった場合でも患者が怪我をしないためにも、息の吹きかけ口を含めて機構全体を軽くする必要がある。

条件 2 の 3 姿勢は図 3.6 の通りである。脊髄損傷患者は 2 時間に 1 回体位を変更する。そのため、患者がどの姿勢であっても対応する機構でなくてはならない。

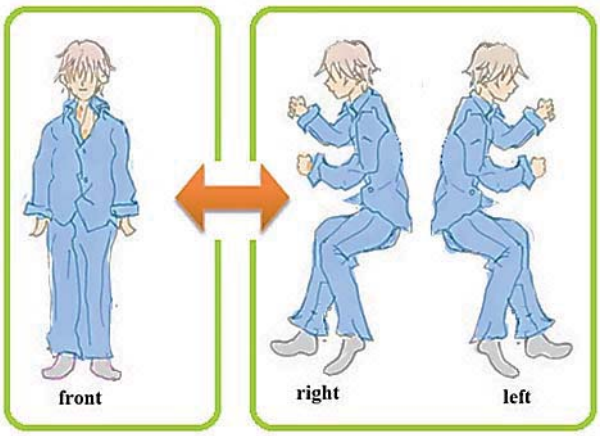


図 3.6：患者の 3 姿勢[50]

3.2.4 機構の構成

息圧センサを接続したチューブを患者の口元付近で固定するための機構は、大きく 3 つの部位に分かれている。ベッドの柵や机に固定する土台である「固定部」、固定部から患者の口元への距離を調節する「調節部」、患者が息を吹きかける「吹きかけ口」である。固定部で机やベッドの柵を挟み込むようにして固定を行い、その上に回転機構が乗る。さらに条件 1、2 に対応する、柔軟な調整を行うことのできるフレキシブルな調節部が伸び、その先端に吹きかけ口が設置される。

第 4 章　タブレットのスリープ解除

4.1 使用するセンサ

本研究で開発するヒューマンインタフェースでは、患者がインターフェースに接触せずに操作できることを目指している。そのため、タブレットのスリープ解除のために息を吹き込む場合も、圧力センサに接続されたチューブを患者が口に銜えない仕様にする。圧力センサに接続したチューブに息を吹きかけて使用するため、直接口に銜えたときの圧力の 20kPa と比べ、格段に下がり 3kPa 以下になると推測される[27,28]。

この圧力帯の仕様を持つセンサとして、Freescale Semiconductor 社の MPXV5004GC7U を使用する。センサは微小な風圧でも、内部の圧電素子によって感知し、それをアンプで増幅し出力している[29]。センサの写真を図 4.1、センサの仕様を表 4.1 に示す[30]。

図 4.1: MPXV5004GC7U[30]

表 4.1: センサの仕様[30]

項目	仕様
型番	MPXV5004GC7U
圧力範囲	0～3.92kPa
最大負荷圧力	16kPa
電源電圧	5V
出力電圧	1.0～4.9V
保存温度	−30～100　℃
動作温度	0～85℃
全体寸法	W:10.79mm,D:11.38mm,H:16.26mm
感圧部寸法	φ3.17
金額（単価）	2000 円

4.2 息圧の測定

前述の通り、センサに接続されたチューブに息を吹きかけた場合の圧力は 3kPa 以下と推測される。しかし息の強さは人によって異なるため、実際の息の圧力（以下：息圧）を測定し、息圧の範囲を知る必要がある。また、実際には多少の圧力損失は存在するものの、理想的にはパスカルの原理によって圧力はチューブの長さには依存しない。測定に使用したオシロスコープは Pico Technology 社 USB オシロスコープ PicoScope2205 である。オシロスコープの写真を図 4.2、仕様を表 4.2 に示す。

図 4.2: PicoScope2205[31]

表 4.2: オシロスコープの仕様[31]

項目	仕様
型番	PicoScope 2205-MSO
帯域幅（バンド幅）	25MHz
サンプリング周波数	1ch:200Ms/s、2ch:100Ms/s
入力端子	BNC コネクタ
電源	USB
本体寸法	100×135×45mm
本体重量	210g

4.2.1 回路の作成

息圧を測定するために回路を作成した。息圧はアナログセンサ（MPXV5004GC7U）を介して電圧値として読み取る（図 4.3）。使用した部品の一覧を表 4.3、回路図を図 4.4、作成した回路を図 4.5 に示す。

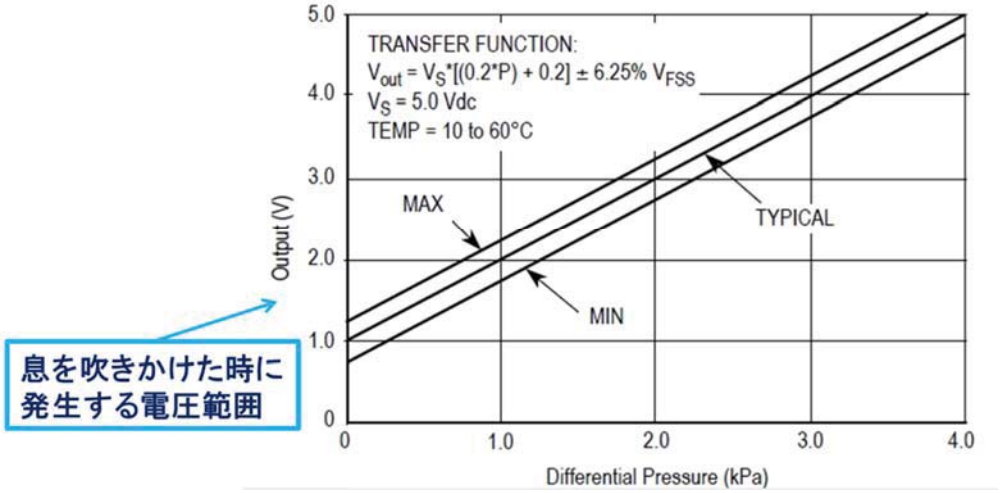


図 4.3: MPXV5004GC7U における圧力と電圧の対応[30]

表 4.3: 使用した部品一覧

部品名	数量（個）
MPXV5004GC7U	1
コンデンサ（470pF）	1
コンデンサ（0.01 μF）	1
コンデンサ（1 μF）	1
抵抗（330 Ω）	1
LED	1

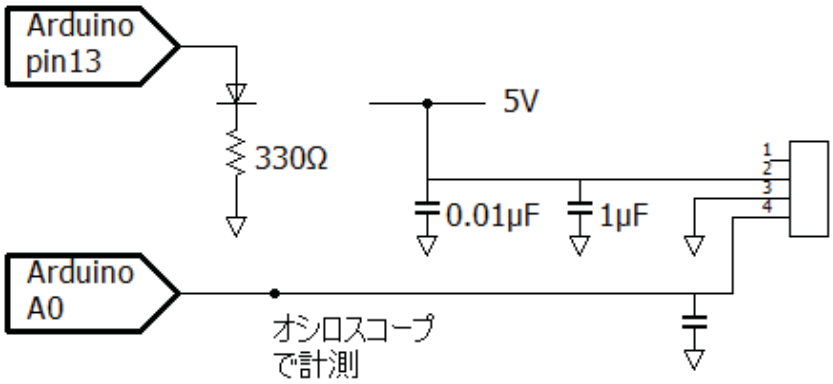


図 4.4: 回路図

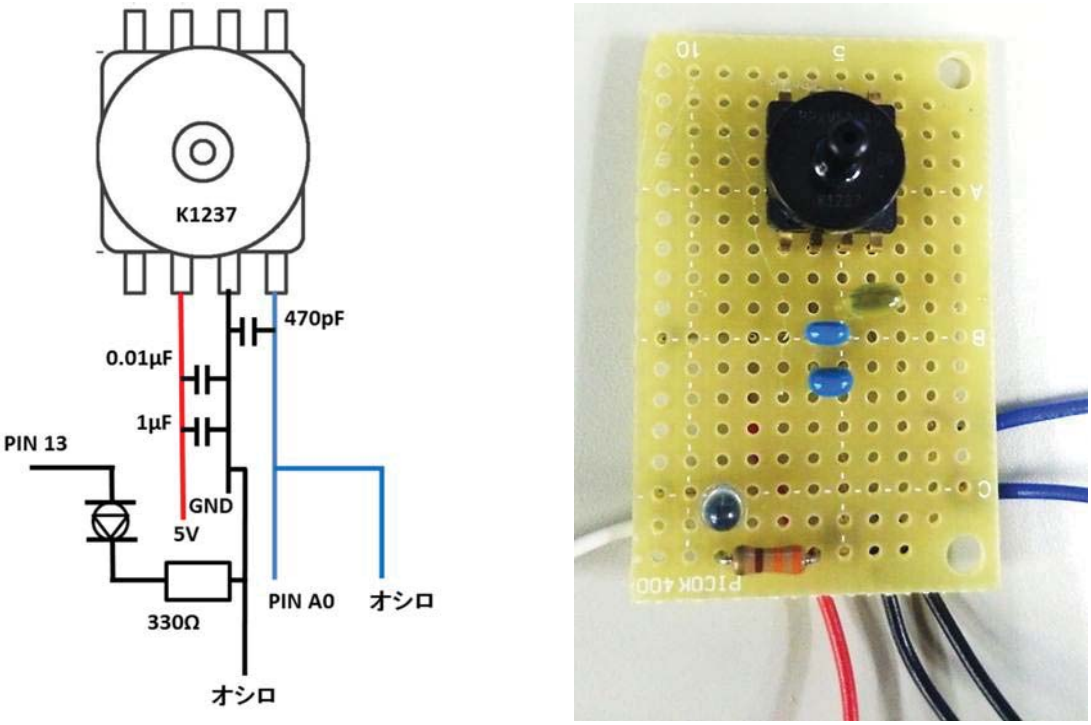


図 4.5: 作成した回路 1

4.2.2 Arduino

息圧センサの値を読み込み、タブレットにスリープ解除の信号を送るために Arduino を用いた。Arduino とは AVR マイコン、C 言語風の Arduino 言語による統合開発環境、入出力（I/O）ポートを備えた基板であり、初心者であっても簡単に扱えるマイコンボードである。

4.3 節にて説明する Android タブレットのスリープ解除では Arduino ADK、4.4 節にて説明する Windows タブレットのスリープ解除では Arduino Leonardo を使用する。Android と Windows では通信方式が異なるため、2 種類の Arduino を使用した。図 4.6 に Arduino ADK を示す。

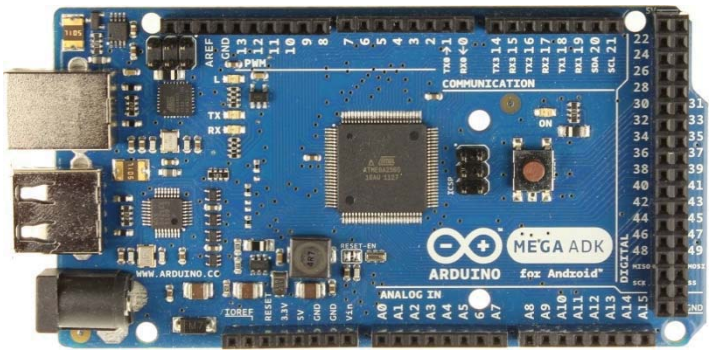


図 4.6: Arduino ADK

4.2.3 実験方法

図 4.7 のように内径 2mm、10mm、30mm のチューブを用意し、図 4.8 のように 50mm 離れ

た位置から息を吹きかける。下記の作業を内径の違う 3 本のチューブで行う。圧力センサの先端部が 3mm のため、太いチューブを使用する際は図 4.9 の様な接続部を使用する。接続部は直径 45.85mm 厚さ 3.9mm の円柱で、片面が塞がれている。そちらを底面として中心に 3mm の穴を空けセンサをはめ込み、反対側の面からチューブを接続した。また、チューブの径を変えて実験する際は 10 分以上間隔を空ける。ただし、実験中はチューブに手を触れないものとする。実験の対象者は 20 代の男性 5 名である。全体図を図 4.10 に示す。

①息を吹き始めてから吹き終わるまでを 1 回と数え、100 回繰り返し息をチューブに吹きかける（※ 1 ）。
②オシロスコープにて息圧センサの電圧の変化を計測する。
③息の音とオシロスコープの波形から、チューブ内に息が入った回数を数える（※2)。
④計測したデータを Excel ファイルに変換し、グラフを作成する。

※1：チューブ内に息が入らなかった場合も 1 回とカウントする。
※2：チューブ内に息が入ると、こもった音がする。息の音とオシロスコープの波形から総合的に判断する。例えば、被験者の吹いた息が弱い場合にオシロスコープの波形が小さくとも、チューブ内に息が入った音がした場合は 1 回と数える。逆に、被験者の吹いた息が強い場合でも、息が入った音がせず、オシロスコープの波形が小さい場合は数えない。

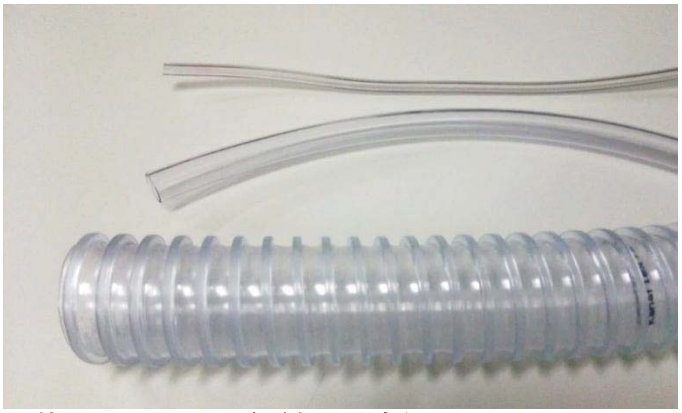


図 4.7: 使用したチューブ（上から内径 2mm、10mm、30mm）

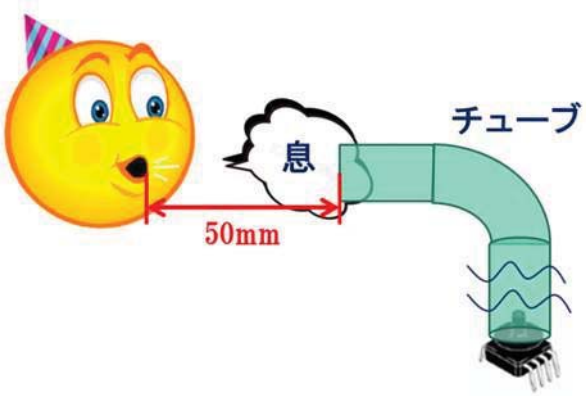


図 4.8: 実験状況図

図 4.9: 接続部

図 4.10: 全体図

4.2.4 実験結果

オシロスコープにて計測した実験結果をグラフにしたものを図 4.11～図 4.13 に示す。このグラフにはある被験者のそれぞれ 100 回の息データが含まれている。図 4.14～図 4.16 に図 4.11～図 4.13 のグラフを一部拡大したものを示す。

（4.1）式より、チューブ内に同じ息（力 F）がかかっているとき、圧力 P を表す電圧はチューブの径（面積 S）が小さくなればなるほど大きくなる。

$$\text{圧力 } \mathbf{P}[N/m^2], [Pa] = \frac{力 F[N]}{面積 A[m^2]} \quad (4.1)$$

グラフより、内径 30mm と内径 10mm のチューブを比較したとき、チューブにかかる圧力は大きくなっていることがわかる。一方で、内径 10mm と内径 2mm のチューブにかかる電圧を比較した場合、内径 30mm のときほど電圧の変化はみられない。これは内径 2mm のチューブが細いため、被験者がふいた息がチューブの正面から外れてしまったためである可能性がある。実際に表 4.4 のように、内径が大きいほど息がチューブに入る回数が多くなる傾向がみられる。また、チューブに息が入った場合でも、径が細いのでチューブの外に息が溢れてしまっていると考えられる。

実験後、被験者に感想を聞いたところ「酸欠で頭がボーとする。」「疲れた」「息切れた」などの意見が出た。このことから、息圧センサの連続使用は患者の身体に負担をかける可能性がある。よって、本研究では患者の麻痺の関係上、体の片側が動かしにくい場合を除き、基本的に息圧センサはタブレットのスリープ解除時のみに用いることにした。

またフリースケール社の仕様書[30]より、このセンサの精度は最大で± 6.25% V_{FSS} である。V_{FSS}=3V より、

$$\pm\ 0.0625 \times 3 = \pm\ 0.1875\ [\mathrm{V}] \quad (4.2)$$

となる。センサに内径 30mm のチューブを用いた場合、計測する電圧も小さくなるためセンサのノイズに影響される場合がある。

以上により、タブレットのスリープ解除には計測される電圧が比較的大きい内径 10mm のチューブを用いることにした。

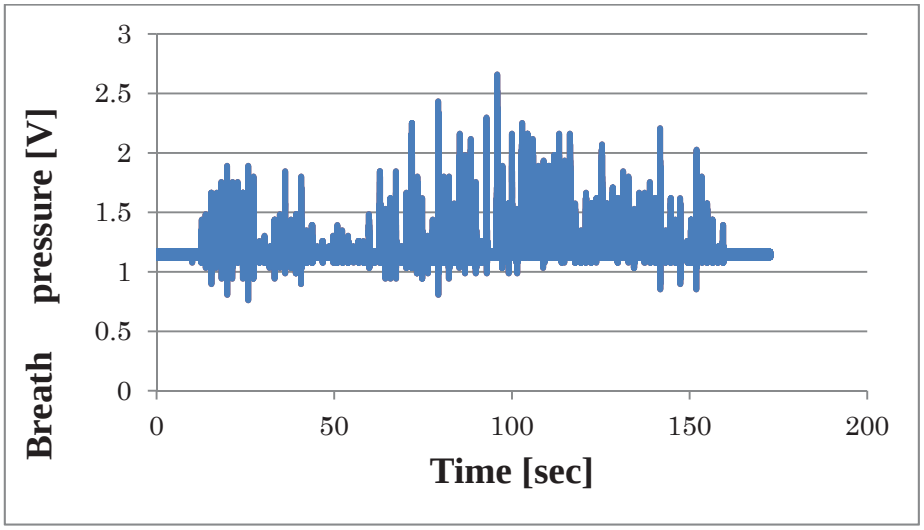


図 4.11:チューブの内径 2mm の息データ（全体）

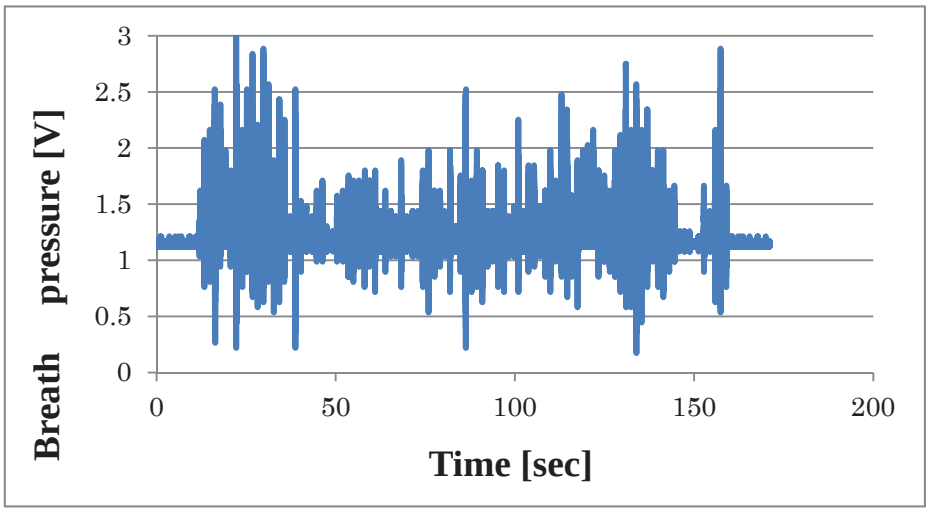


図 4.12: チューブの内径 10mm の息データ（全体）

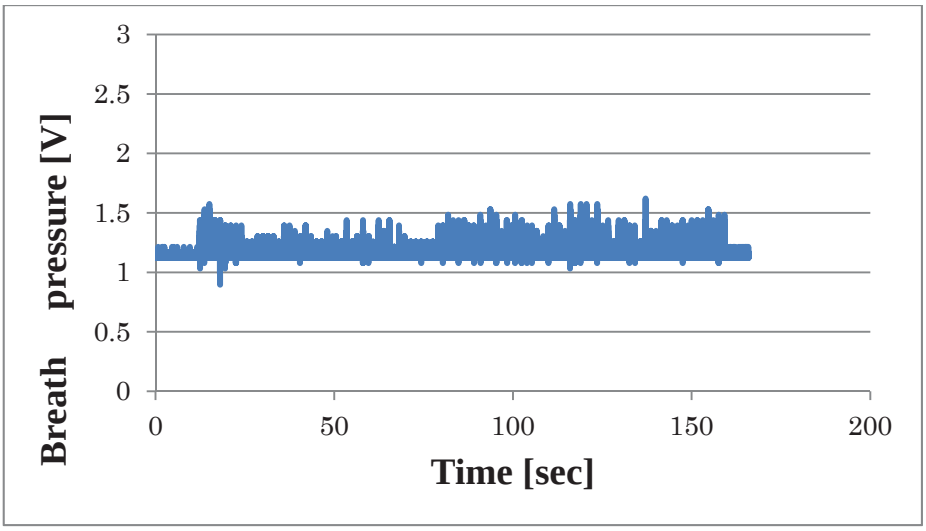


図 4.13: チューブの内径 30mm の息データ（全体）

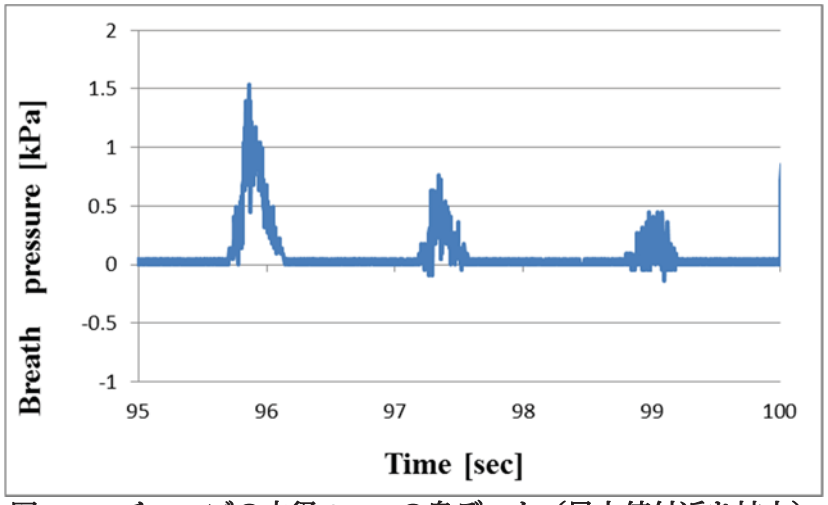


図 4.14: チューブの内径 2mm の息データ（最大値付近を拡大）

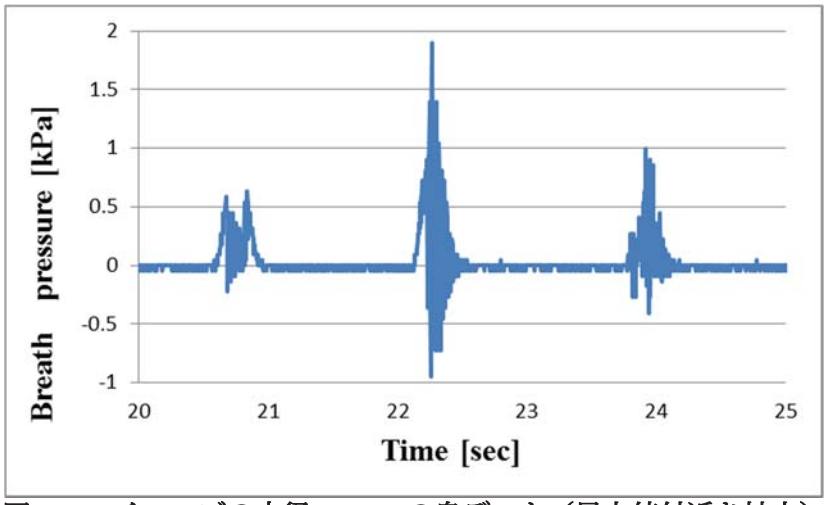


図 4.15: チューブの内径 10mm の息データ（最大値付近を拡大）

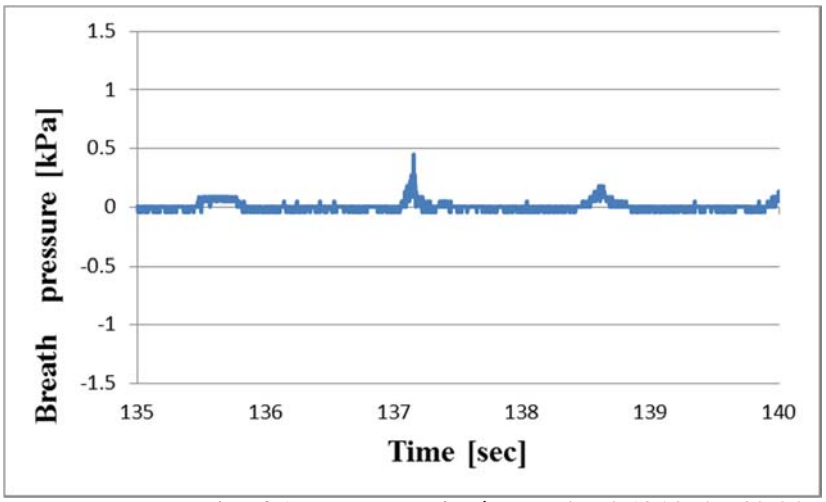


図 4.16: チューブの内径 30mm の息データ（最大値付近を拡大）

表 4.4: 息が入った回数

	息が入った回数（回）				
チューブの内径（mm）	被験者 1	被験者 2	被験者 3	被験者 4	被験者 5
2mm	85	75	74	84	73
10mm	96	94	85	88	97
30mm	95	100	97	77	99

4.2.5 息判定の仕様

実験を行ったところ、推測通り息圧の大きさには個人差があることがわかった。実験で得られたデータを元に閾値を決め、一定時間内に規定の回数閾値を超えた場合に Arduino からタブレットに信号を送り、タブレットのスリープを解除する。想定するインターフェースの使用者には高齢者も含まれるため、息圧も小さくなることが予想される。そこで息圧が最も小さかった人のデータを元に閾値を 1.21V、閾値を数える時間窓を 1500ms、その時間内に 100ms ごとに息圧を調べたときに閾値を超えるべき回数を 3 回と定めた（図 4.17）。

また、閾値の電圧 1.21V を圧力に変換すると

$$\mathbf{Vout} = \mathbf{(V/P)} * \mathbf{P} + \mathbf{V}_{\mathbf{off}} \qquad \mathbf{(4.3)}$$

V/P=1.0、V_{off} = 1.17V より、

$$Vout = P + 1.17$$

変形すると

$$\mathbf{P = Vout - 1.17\ [kPa]} \qquad \mathbf{(4.4)}$$

となる。閾値 1.21V を式 4.2 に代入すると、息圧は、

$$P = 1.21 - 1.17$$
$$= 0.04[kPa] \quad = 40[Pa]$$

である。内径 10mm のチューブにて、電圧を圧力に変換した息圧が最も小さかった人の息データを図 4.18、一般的な息圧のデータを図 4.19 に示す。また、実験の被験者 5 名の息の電圧を圧力に変換したものを付録に記す。

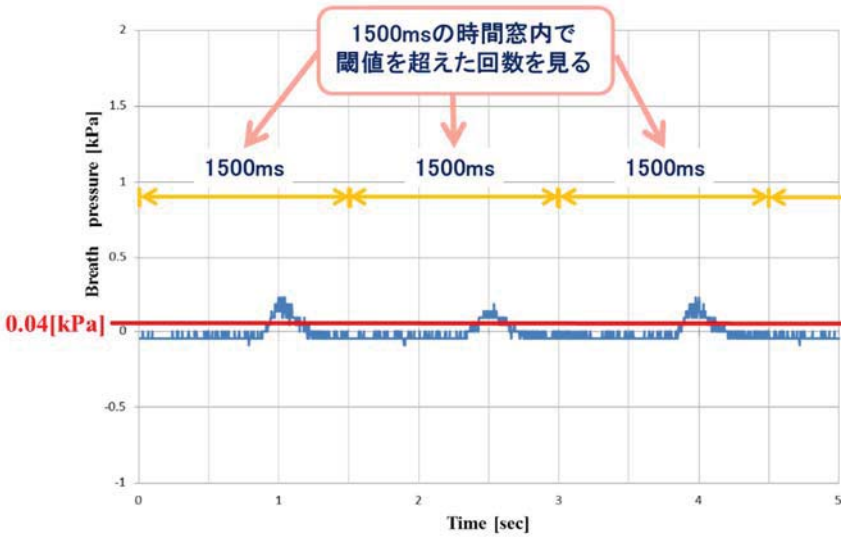


図 4.17: 息判定の仕様

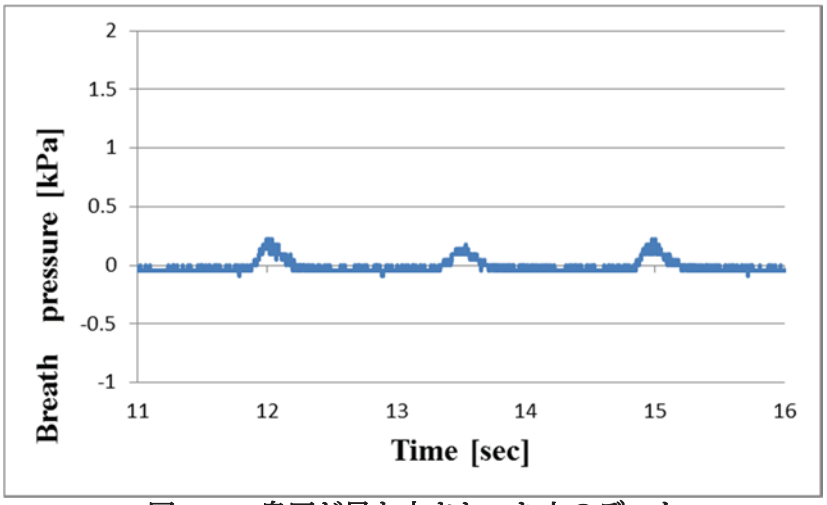


図 4.18: 息圧が最も小さかった人のデータ

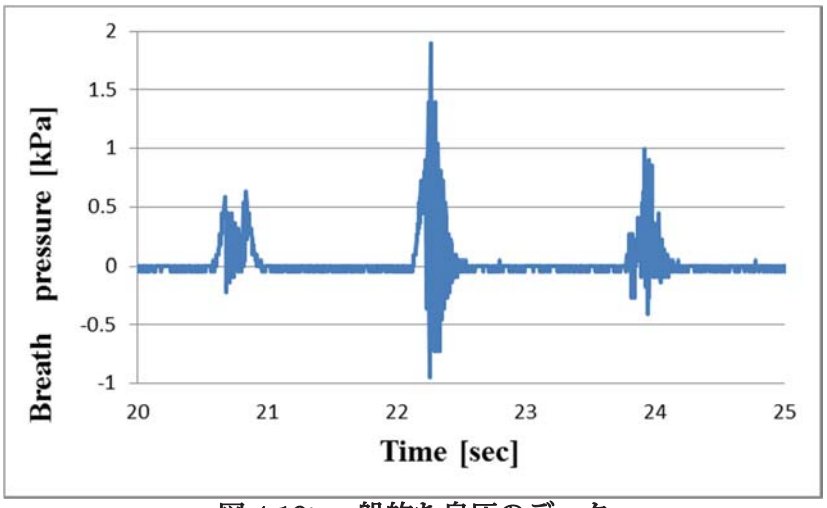


図 4.19: 一般的な息圧のデータ

4.3 息圧による Android タブレットのスリープ解除

この研究を始めた当初 Windows タブレットは発売されたばかりであり、タブレットは Android と iOS が主流であった（図 4.20）。また、iOS 用アプリケーションの開発には Mac OS X が必要であり、開発言語に Objective -C を使うため、敷居が高い。一方で Android は Windows、 Mac、Linux など開発の環境を選ばず、Java 言語を使用し、かつ開発環境も無償なので敷居が低いため初心者でも開発が行いやすい。よって、まずプロトタイプとして Android タブレットを用いてスリープの解除を行うことにした。

一方で図 4.21 のように、パソコンの OS は Windows が 92％と圧倒的なシェアを誇っている[47]。よって本研究では、Android タブレットにてプロトタイプ作成後、Windows タブレットにてヒューマンインタフェースを作成することにした。

Android タブレットの開発環境は Eclipse、言語は Java を使用する。また、息圧を読み取るための Arduino の開発環境には Arduino IDE、言語は C を使用する。

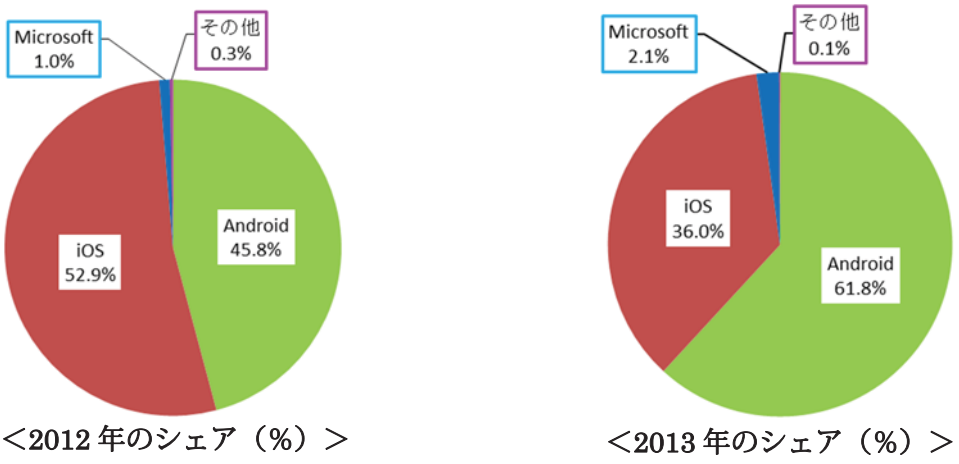


図 4.20: OS 別世界タブレット販売ランキング（2012 年、2013 年）[32]

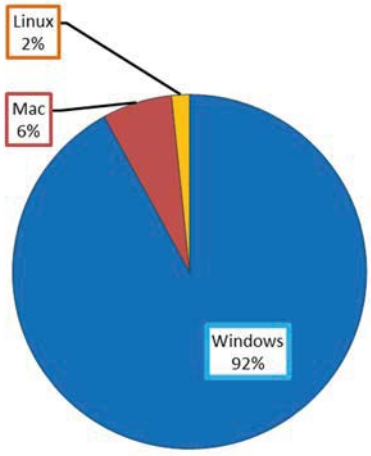


図 4.21: デスクトップ OS シェア（2014 年 9 月）[47]

4.3.1 Arduino による息の判定プログラム

息がセンサに吹きかけられたかどうかの判定を Arduino で行う。そのために次のような流れのアルゴリズムを組む。初めに図 4.22 のようにセンサのアナログ値を 100ms おきに読み込む。

```
// アナログ値を読み込む
status = analogRead(0);
```

図 4.22: センサのアナログ値を読み込む

次に、図 4.23 のような息圧の判定を行う。判定の条件は、息圧センサの出力電圧が時間窓 1500ms 以内に閾値 1.21V を 3 回超えることであり、条件を達成するまでループを繰り返す。

```
if (status >= 247){
        /*
        閾値 ： 測定電圧 1.21（V）÷基準電圧（電源電圧）（5V）
                        × 1023（変換段階数） ＝ A/D 変換出力値（閾値）247.7
        */
         //閾値が 2 回以上超える
         if (kaisuu < 2) {
            kaisuu=kaisuu+1;
         }else{  // 閾値より大きくなった&2 回以上なら以下を処理
         //タブレットの起動
         }
}
```

図 4.23:　息圧の判定

またセンサで取得した値がアナログ値のため、閾値の判定の際には A/D 変換後のデジタルの値を用いる。A/D 変換後の閾値を求める式を(4.5)式に示す。

$$\begin{aligned}\textbf{A/D 変換出力値(閾値)} &= \textbf{測定電圧(V)} \div \textbf{基準電圧(電源電圧)} \times \textbf{1023(変換段階数)} \\ &= \textbf{1.21（V）} \div \textbf{5V} \times \textbf{1023} \\ &= \textbf{247.7}\end{aligned} \qquad \textbf{(4.5)}$$

条件を満たしたら図 4.24 のようにタブレットに信号を送る。

```
// 読み込んだアナログ値を byte 配列に格納
msg[0] = (status >> 8) & 0xff;
msg[1] = status &0xff;

// Android に送信
acc.write(msg, sizeof(msg));
```

図 4.24: Android へのデータ送信

同時に、作成した回路（図 4.4） の LED を 1 秒間点灯させる。なお、時間窓の 1500ms を超えると閾値を超えたカウントのリセットを行う。また、息の判定のしくみについて図 4.25 に示す。実際に使用したプログラムの全文は付録に記す。

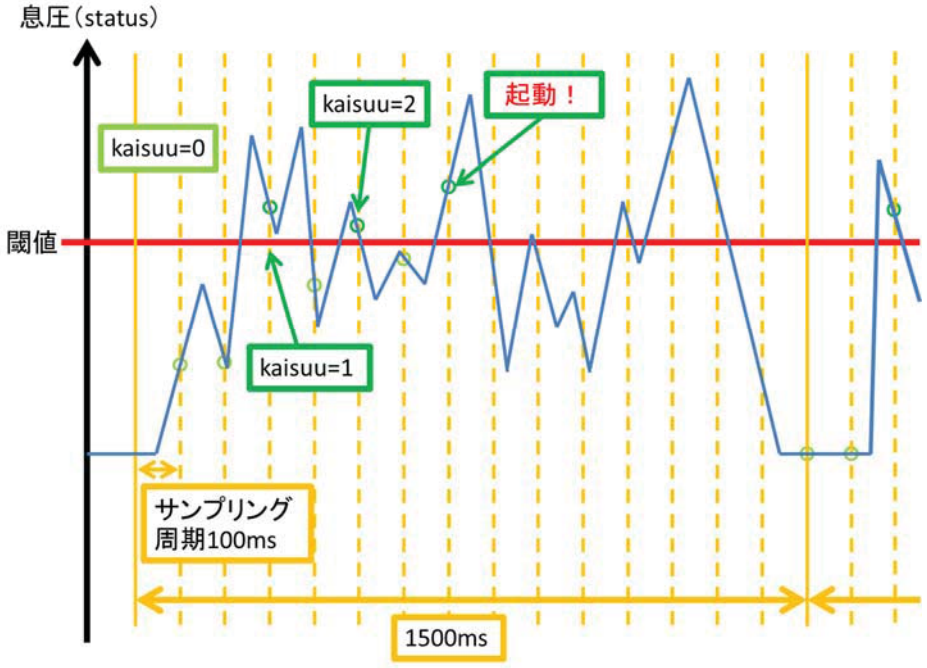


図 4.25： 息の判定 2

4.3.2 Android タブレットのスリープ解除プログラム

Android タブレットでは、USB を通して ADK という仕組みで Arduino と通信を行う。そして図 4.26 のように、Arduino から信号が送られてきた場合に Android タブレットをスリープ状態から復帰させ、スクリーンロックを解除する。解除後アプリを起動させて、画面上に Status ON とテキストを表示する。

Android タブレットのスリープ解除プログラムの全文を付録に示す。また、Android タブレットのスリープ解除の様子を図 4.27 に示す。

```
//スマホの電源確保、スリープ状態から復帰する
            wakelock = ((PowerManager)
            context.getSystemService(Context.POWER_SERVICE))
              .newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK
              | PowerManager.ACQUIRE_CAUSES_WAKEUP
              | PowerManager.ON_AFTER_RELEASE, "disableLock");
              wakelock.acquire();


          //スクリーンロックを解除する
KeyguardManager km = (KeyguardManager)
                    context.getSystemService(Context.KEYGUARD_SERVICE);
          KeyguardManager.KeyguardLock klock = km.newKeyguardLock("disableLock");
```

```
klock.disableKeyguard();

//アラームを実行する MainActivity.java へ
Intent intent1 = new Intent(context, MainActivity.class);
intent1.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(intent1);
```

図 4.26:　スリープ状態から復帰、スクリーンロックを解除

図 4.27: Android タブレットのスリープ解除の様子

4.4 息圧による Windows タブレットのスリープ解除

プロトタイプである Android タブレットのスリープ解除を行うことができたため、次に Windows タブレットにおけるスリープ解除を実現する。Windows タブレットにおけるスリープ解除では Arduino Leonardo を用いて、マウスクリックに相当する信号を送信することでスクリーンセーバーの解除を行う。Arduino Leonardo については次節にて説明する。

4.4.1 Arduino Leonardo

Arduino Leonardo とは、ATmega32u4 チップを搭載した Arduino 純正ボードである（図 4.28）。他の Arduino とは異なり、キーボードやマウスとして振る舞うことができる[33]。

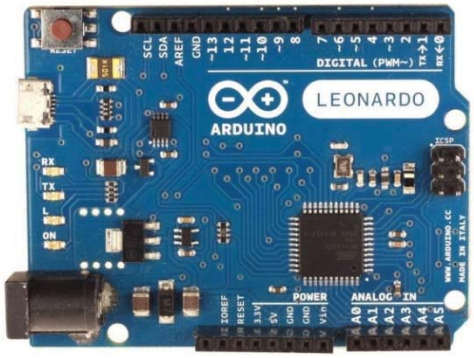


図 4.28: Arduino Leonardo[34]

4.4.2 Arduino による Windows タブレットのスリープ解除プログラム

Windows タブレットのスリープ解除では Android タブレットの時とは異なり、シリアル通信を使用しない。また、Arduino Leonardo をマウスとして使用するため、マウスライブラリを使用する宣言をする（図 4.29）。

```
void setup()//最初に一度だけ実行される部分
{
   pinMode(0, INPUT); // A0 番ピン圧力センサからの入力に設定
   pinMode(13, OUTPUT); // 13 番ピンを出力に設定

   Mouse.begin(); //マウスライブラリを開始する
}
```

図 4.29:　マウスライブラリの起動

Arduino 上で息圧の判定後、図 4.30 のようにマウスクリックの命令を Windows タブレットへ送ることで、Windows タブレットのスリープ解除を行う。これは Android タブレットの図 4.39 に対応している。

```
if(msg[0] != previous && msg[0]==1){
//msg と previous が等しくない(!=)、かつ previous0⇒1

      Mouse.click(); //左クリックを送信する。
}
```

図 4.30: マウスクリックのプログラム

息をチューブへ吹き込んだところ、スリープを解除できた。Arduino を含む回路と Windows タブレットの配線を図 4.31、Arduino による Windows タブレットのスリープ解除プログラムの全文を付録に示す。

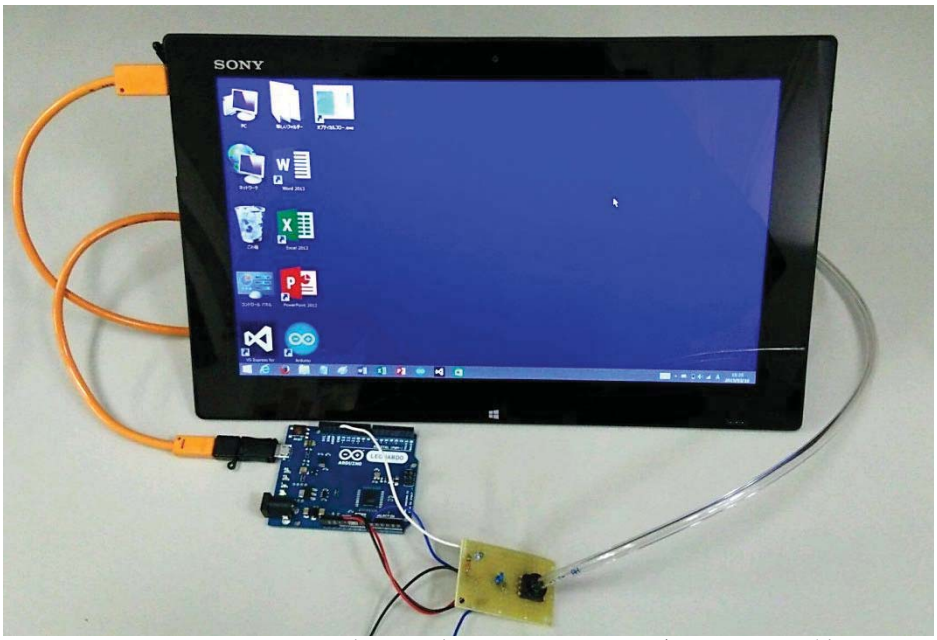


図 4.31: Arduino を含む回路と Windows タブレットの配線

第 5 章 機構

息圧センサを接続したチューブを患者の口元付近で固定するための器具を作成する。器具は大きく 3 つの部位に分かれており、ベッドに固定する土台である「固定部」、固定部から患者の口元への距離を調節する「調節部」、患者が息を吹きかける「吹きかけ口」である。固定部で机やベッドの柵を挟み込むようにして固定を行い、その上に回転機構が乗る。さらに、チューブ内に太さ 4mm の針金を通すことでフレキシブルに調整が行うことのできる調節部が伸び、その先端に吹きかけ口が設置される。

吹きかけ口は内径 10mm のチューブを 3 本まで設置することができる。1 本は息圧によるタブレットのスリープ解除に用いるが、残り 2 本は万が一、一定方向のみ首が動かしづらい場合に、その向きのカーソル操作を画像処理の代わりに息圧センサで代用するためのものである。第 6 章にて、息圧センサによるカーソル操作について述べる。吹きかけ口から伸びたチューブは調節部の中を通り、息圧センサに接続されている。

各部位の自由度は、固定部が 1 自由度、調節部が 6 自由度、吹きかけ口が 1 自由度である。本体の高さは最長 80cm、幅は 12cm、総重量は 825 g であり、軽量である。固定部を図 5.1、調節部を図 5.2、吹きかけ口を図 5.3、機構全体とタブレットとの配置関係を図 5.4 に示す。

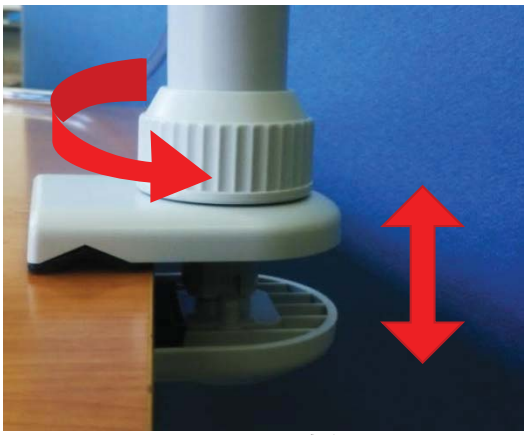


図 5.1: 固定部

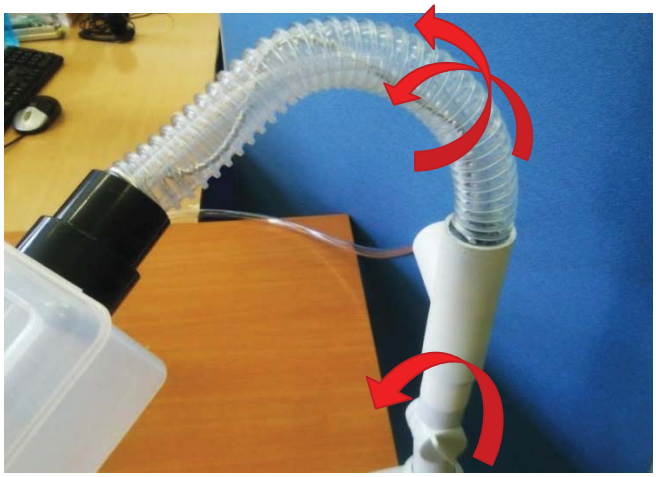


図 5.2: 調節部

図 5.3: 吹きかけ口

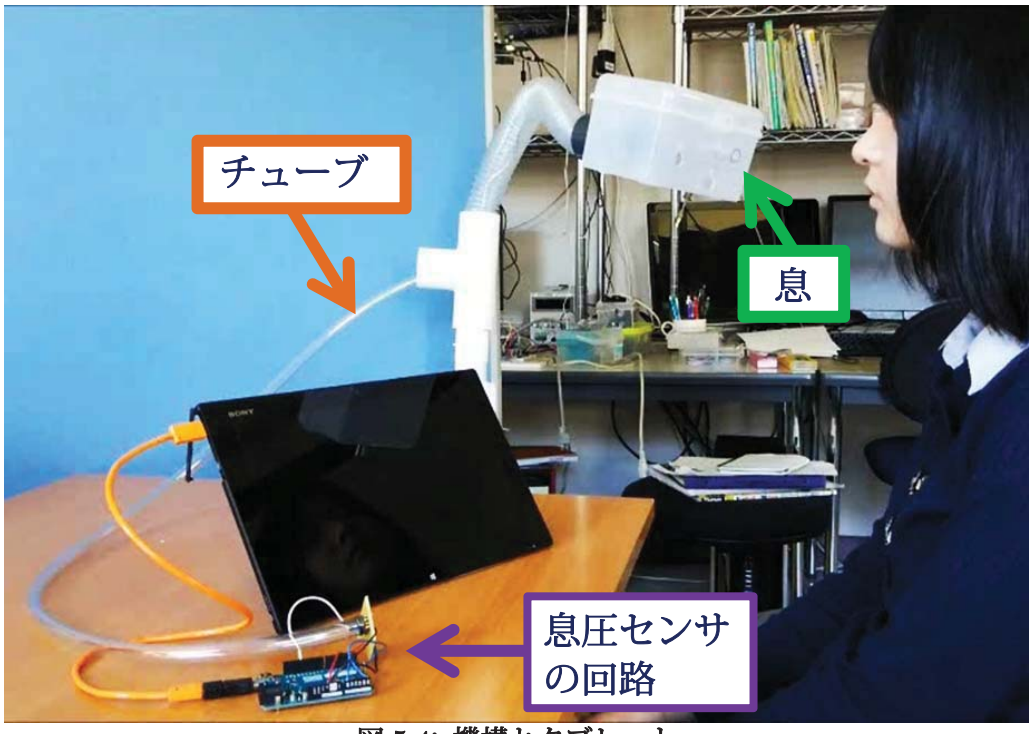


図 5.4: 機構とタブレット

第 6 章 息圧によるマウスカーソルの操作

マウスカーソルの移動は第 7 章で解説する画像処理による顔の位置を用いる。しかし、患者が一定方向に顔を動かしづらい場合に、その向きのカーソル操作を息圧センサで代用することを考えた。そのために、息圧を用いて、Windows タブレットにおけるマウスクリックと上下左右の 4 方向へカーソルの移動を可能にする。

6.1 タクトスイッチによるマウカーソルの操作

6.1.1 タクトスイッチによるプロトタイプの作成

マウスクリックと上下左右の 4 方向へカーソルの移動のために、本章では息圧センサを 5 つ使用する。息圧センサが 1 つのとき 4 章と比べて操作が複雑になるため、プロトタイプとしてブレッドボード上にタクトスイッチを配置し、動作確認用の簡易的な回路を作成した。使用した部品の一覧を表 6.1、ブレッドボード上の簡易回路を図 6.1 に示す。

表 6.1: 使用した部品一覧

部品名	数量(個)
タクトスイッチ	5
抵抗(330Ω)	5
抵抗(1KΩ)	5
LED	5

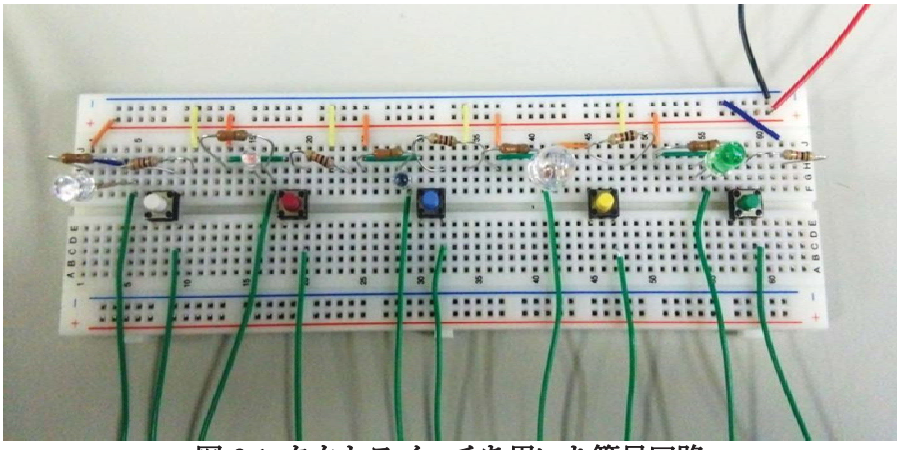


図 6.1 タクトスイッチを用いた簡易回路

6.1.2 タクトスイッチによるマウカーソルの操作プログラム（Arduino）

この回路は 4 つのタクトスイッチが、上下左右へのマウスの移動に対応している。タクトスイッチを押すとカーソルが移動を始め、もう一度同じスイッチを押すと止まり、他方向への移動スイッチを押すと押した方向へ移動し始めるよう Arduino のプログラムを作成する。さらにクリックと対応するタクトスイッチが押された場合はその場所でマウスクリックを行う。

まず、図 6.2 のようにタクトスイッチからデジタル信号を読み取る。

```
// スイッチの状態を読み取る
int val_up = digitalRead(BUTTON_up);
int val_down = digitalRead(BUTTON_down);
int val_left = digitalRead(BUTTON_left);
int val_right = digitalRead(BUTTON_right);
int val_click = digitalRead(BUTTON_click);
```

図 6.2: タクトスイッチの状態を読み取る

次に図 6.3 のように上に対応するタクトスイッチが押されており、かつその前にそのスイッチが押されていなかったときに前回のスイッチが押されていたか否かの情報ををを示すフラグ（G_state_up）の切り替えを行う。そして、スイッチの状態を 1 つ前の情報として保存する。

```
//スイッチが ON、前回が OFF だったとき
if(val_up == SW_ON && G_oldval_up == SW_OFF) {
   //LED 点滅の状態を切り替える
      if(G_state_up == FLG_ON){      // フラグが ON のとき
         G_state_up = FLG_OFF;            // OFF
      } else {                          // フラグが OFF のとき
         G_state_up = FLG_ON;             // ON
      }
}
G_oldval_up = val_up;                  // スイッチの状態を G_oldval_up へ保存
```

図 6.3: フラグの切り替え

フラグが ON のとき、マウスカーソル移動を行う。図 6.4 では、上向きに 1 ピクセルずつ移動を行っている。移動量を変化させるには、数値を変えればよい。例えば現在の 10 倍早くカーソルを上に移動したい場合は、y の値を－1 から－10 にする。

```
//マウスの移動を開始するとき
if (G_state_up == FLG_ON){          // フラグが ON のとき
      digitalWrite(LED_up, SW_ON);      // 緑色 LED 点灯
      //マウス上にを動かす　(x, y, wheel マウスホイールをスクロールする量)
      Mouse.move(0,  －1, 0);
}
```

図 6.4: 上方向へのマウスカーソルの移動

さらに、他のスイッチを切り、フラグのリセットを行う。上方向と同様にして下方向、左方向、右方向の移動を行う。

　マウスクリックに対応するスイッチの場合は、図 6.5 のようにマウスクリックの関数を実行し、100 ミリ秒後 LED を消灯する。その後はカーソル移動のときと同様に、他のスイッチを切り、フラグのリセットを行う。

```
if (G_state_click == FLG_ON){            //フラグが ON のとき
      digitalWrite(LED_click, SW_ON);      //  赤色 LED 点灯
      Mouse.click(); //左クリックを送信する

      delay(100);
      digitalWrite(LED_click, SW_OFF);
}
```

図 6.5: 左クリックを Windows タブレットへ送信

カーソル移動のタクトスイッチを押すことで、任意の方向へのカーソル移動と移動方向の切り替えを行うことができた。また、マウスクリックに対応するタクトスイッチが押された場合には、その場所でマウスクリックを行えることを確認した。タクトスイッチによるマウカーソルの操作プログラムの全文を付録に示す。

6.2 息圧センサによるマウスカーソル操作

6.2.1 5 つの息圧センサを用いた回路の作成

5 つのタクトスイッチを用いた回路（図 6.1）と図 4.4 の回路を参考に、圧力センサを 5 つ用いた回路を作成した。使用した部品の一覧を表 6.2、配線図を図 6.6、作成した回路を図 6.7 に示す。また図 6.8 のように、回路の右上にあるメスピンにリード線を差し込むことにより、オシロスコープで息圧を測定することができる。オスピン L 字が GND に繋がっており、残りのメスピンは上からそれぞれ上・下・左・右・クリックのセンサに繋がっている。

表 6.2: 使用した部品の一覧

部品名	数量（個）
MPXV5004GC7U	5
コンデンサ（470pF）	5
コンデンサ（0.01 μF）	5
コンデンサ（1 μF）	5
抵抗（330 Ω）	5
LED	5
メスピン（6 × 1）	1
オスピン L 字（1 × 1）	1
オスピン（5 × 1）	2
オスピン（2 × 1）	1

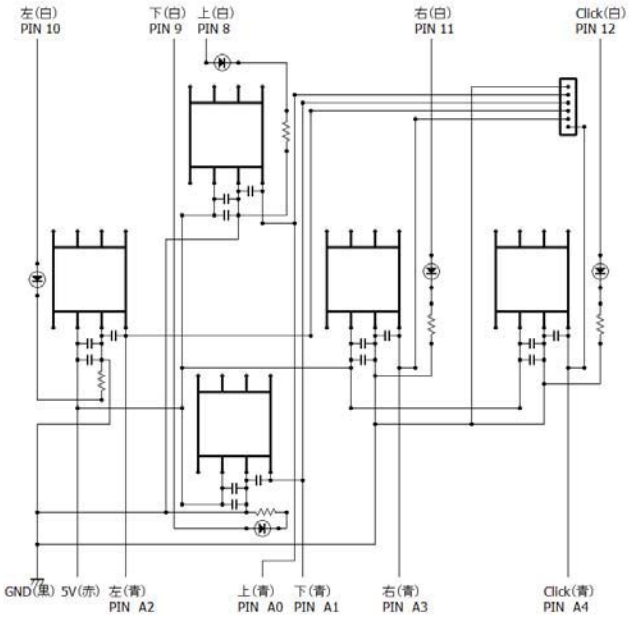
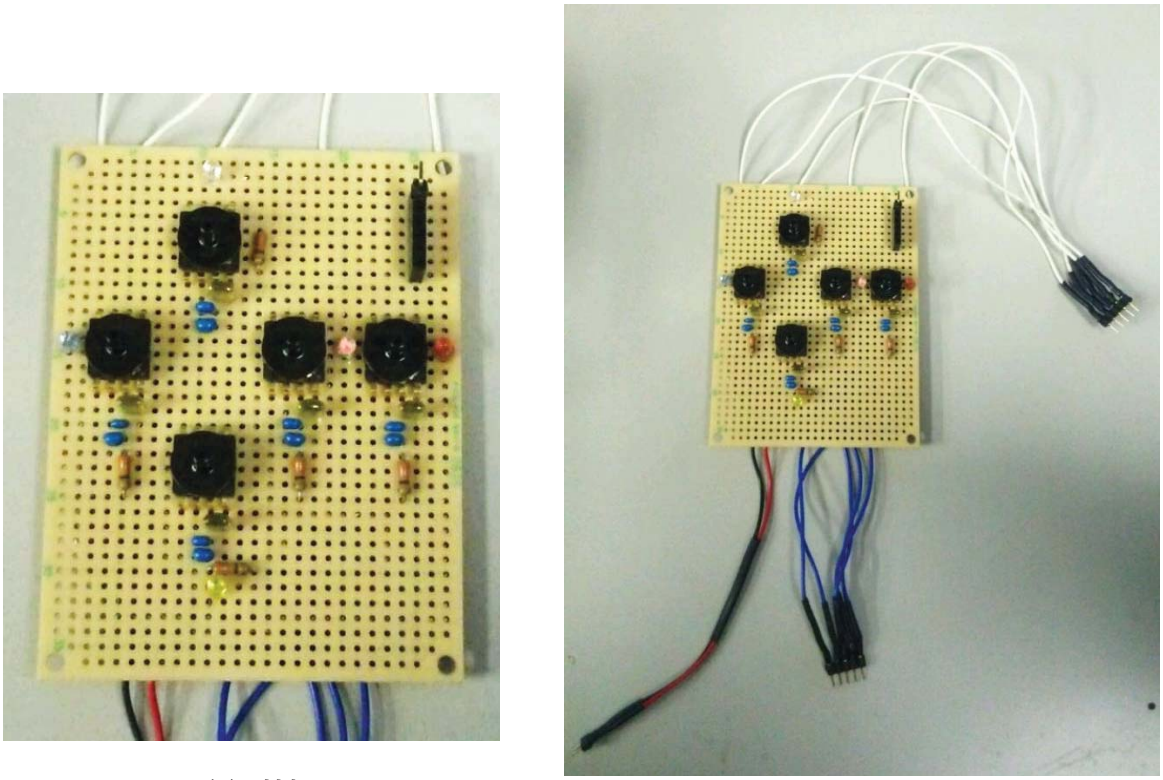


図 6.6: 配線図



（a）基板　　（b）全体

図 6.7: 作成した回路

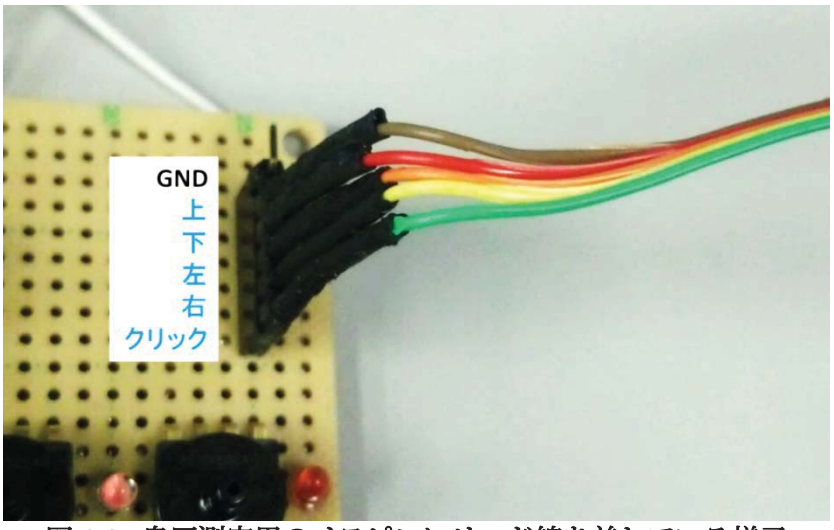


図 6.8: 息圧測定用のメスピンにリード線を差している様子

6.2.2 息圧センサによるマウカーソルの操作プログラム（Arduino）

前述した「4.4.2 Arduino による Windows タブレットのスリープ解除プログラム」と「6.1.2 タクトスイッチによるマウカーソルの操作プログラム」を組み合わせることで、5 つの息圧センサによるマウカーソルの移動プログラムを作成する。

圧力センサごとに息圧の判定を行い、息圧を検知したらフラグを ON にして命令を Windows タブレットに送信し、その後フラグを OFF にする。他のセンサに反応があった場合も同様である。息圧センサによるマウカーソルの移動プログラムの全文を付録に示す。

6.2.3 息圧センサによるマウスカーソル操作の問題点

息圧センサによるマウスカーソル操作を実際に試してみたところ、大きく分けて以下の 2 つの問題点が挙げられた。

【問題点 1】長時間、使用する際に息が切れる。
息圧センサは短時間の使用は問題ないが、操作に息を使い続けることにより、息切れを起こす可能性がある。このヒューマンインタフェースの使用者には高齢者や長期の闘病生活によって体力の低下した患者も含まれるため、患者に負担をかける仕様は好ましくない。

【問題点 2】息圧センサの吹きかけ角度と画面の角度が異なる。
息圧センサを吹きかけるのに丁度良い角度や位置と、タブレットの画面を見るのに丁度良い角度や位置は必ずしも同じではない。そのため、操作画面と息圧センサの吹きかけ口の間で視点を移動しなくてはならず、操作にタイムラグが生まれてしまう。

本章の息圧センサによるマウスカーソル操作と比較して、第 7 章の画像処理によるマウスカーソル操作では、顔を傾けるだけで操作が可能なため長時間の使用でも患者の負担にならないと考えた。また、タブレットの画面内で操作が行えるため、視点の移動による操作時のタイムラグが殆どない。一方で、息圧センサは Windows タブレットがスクリーンセーバーの状態でも、外部から操作できるため、タブレットのスリープ解除など使用頻度の少ない動作のインターフェースには向いていると考える。

第 7 章　画像処理によるタブレットのカーソル操作

7.1 顔の移動によるマウスカーソルの操作

タブレット上でマウスカーソルの移動を行うために、図 7.1 のようにタブレットのカメラから映像を取得し、画像処理の手法の 1 つであるオプティカルフローを用いて顔の移動方向と距離を計算する。まず本節では顔の移動に応じて上下左右へマウスカーソルの移動を行う。本研究の対象である患者はベッドで多くの時間を過ごすため、患者の背後の物や人が移動することは、殆どない。よって本研究では画面内の動きは患者の動きであると考えることができる。

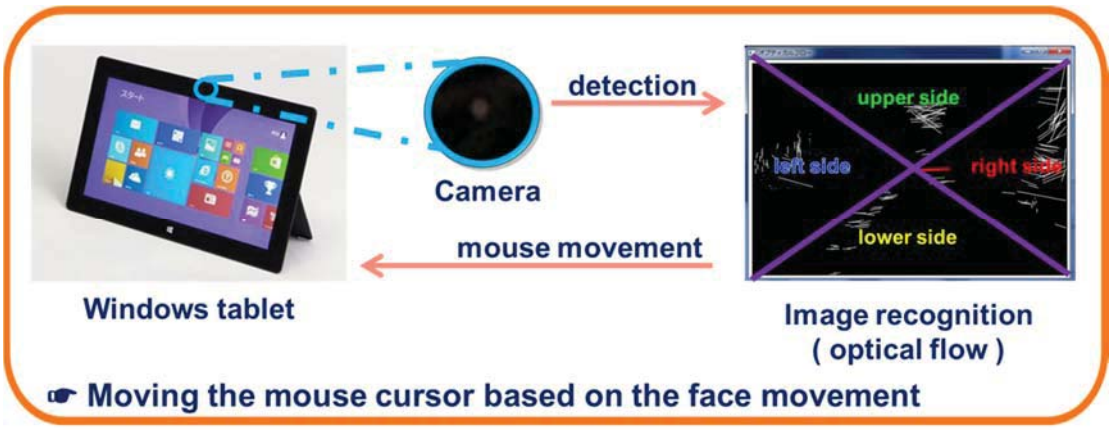


図 7.1: 顔の移動によるマウスカーソルの操作

7.1.1 OpenCV

本章では画像処理ライブラリである OpenCV を用いて物体の移動や顔を検出し、検出した結果を元にマウス操作を行う。また、第 4 章のスリープ解除では Android タブレットと Windows タブレットの両方を用いたが、これ以降は Windows のみを使用する。これは、画像処理ライブラリである OpenCV は Android と比べて Windows の方が処理が高速であり、取り扱いが容易であるためである。

OpenCV（Open Source Computer Vision Library）とは、1998 年に Intel が開発を始め、現在は Willow Garage が引き継いで活発に開発を行っている無償のオープンソース画像処理ライブラリ集である。OpenCV には、画像のフィルタ処理やテンプレートマッチング、物体認識、映像解析などのアルゴリズムが 500 以上用意されている[36]。

まず、初めはデスクトップパソコンと Web カメラを用いて、プログラムの動作確認を行う。最後に Windows タブレットに統合を行う。開発環境は Microsoft Visual C++ 2008 Express Edition を使用する。

7.1.2 特徴点抽出の理論

映像内の物体を検出する手法として、物体の形状を見つけるテンプレートマッチングなどが挙げられる。しかし、この手法は物体の向きが変われば、テンプレートがマッチしないという問題がある。そこで、物体の特徴的な箇所だけに注目をして検出を行う特徴点抽出を用いる。特徴点には画像中の線の端点（線の終点）、交差点（画像の境目や角）、色彩が大きく変わる濃淡の濃い部分や輪郭などが用いられる[35]。

このトラッキングに適した特徴は、2 直線の交わる場所で微分値が大きい場所にあることからコーナーと呼ばれる。2 直線の交点を見る理由は、微分値が大きい点はエッジ上にある場合に同じエッジに沿った別の点と似ている可能性があり、判断がつきにくいからである。例えば、図 7.2 の円で囲まれた点はトラッキングに適しているが、四角の中の点ははっきりとしたエッジがあるにも関わらず、トラッキングに適していない。

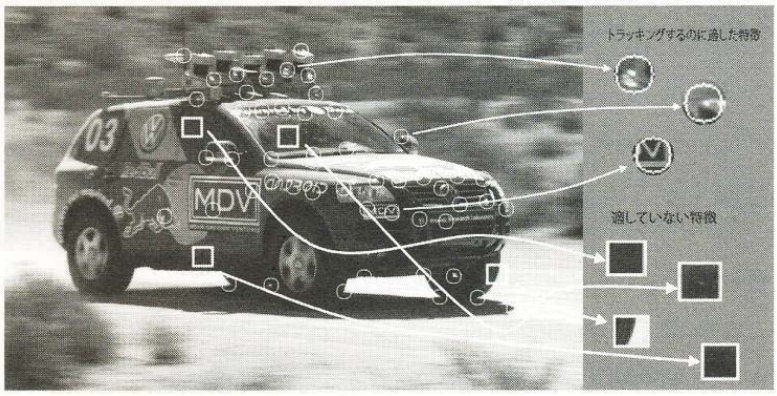


図 7.2: トラッキングに適した点[46]

コーナーの点は、連続フレーム間で対応するものを見つけ出すために十分な情報を含んでいる点であり、これを特徴点として採用する。

最もよく使用される Harris の手法では、画像の彩度を 2 次微分した行列を用いる。画像内の全ての点に対して 2 次微分を行い、新しい複数の 2 次微分画像もしくは、両方を一緒にした Hessian 画像を形成する。Hessian 画像はある点（p）の周りの Hessian 行列に由来し、(5.1)式の 2 次元行列で定義する。

$$\mathbf{H}(\boldsymbol{p}) = \begin{bmatrix} \dfrac{\partial^2 I}{\partial x^2} & \dfrac{\partial^2 I}{\partial x \partial y} \\ \dfrac{\partial^2 I}{\partial x \partial y} & \dfrac{\partial^2 I}{\partial y^2} \end{bmatrix}_{\boldsymbol{p}} \qquad (5.1)$$

Harris コーナーでは、各点の周りにある小さなウィンドウ上における 2 次微分画像の自己相関行列を考える。画像 I の x 方向微分を Ix、y 方向微分を Iy としたとき、行列は(5.2)式のように定義される。

$$\mathbf{M}(\mathbf{x},\mathbf{y}) = \begin{bmatrix} \sum_{-K\le i,j\le K} w_{i,j} I_x^2(x+i,y+j) & \sum_{-K\le i,j\le K} w_{i,j} I_x(x+i,y+j) I_y(x+i,y+j) \\ \sum_{-K\le i,j\le K} w_{i,j} I_x(x+i,y+j) I_y(x+i,y+j) & \sum_{-K\le i,j\le K} w_{i,j} I_y^2(x+i,y+j) \end{bmatrix} \qquad (5.2)$$

ここで$w_{i,j}$は、重みづけの項であり同じ値にすることも可能であるが、丸いウィンドウや Gaussian 重み付けを行うのによく使われる。Harris の定義によると、コーナーは 2 次微分の自己相関行列が 2 つの大きな固有値を持つ画像内の場所である。そのような点を中心として少なくとも別の 2 方向へ向かうテクスチャ（またはエッジ）が存在する。2 次微分は勾配が一定（定数）だと 0 になるため、反応しないメリットがある。さらに、自己相関行列の固有値だけを考える場合には、回転に対しても不変な量を考えている。これはトラッキングの対象が移動や回転を行うため重要である。固有値は特徴点にもなる。

後に発見された Shi と Tomasi の手法では、2 つの固有値のうち小さい方が閾値よりも大きければよいコーナーとなる。Shi と Tomasi の方法は、Harris の方法よりも優れた結果を出す。次節で使用する cvGoodFeaturesToTrack 関数は、この Shi と Tomasi の手法を実装している[46]。

7.1.3 特徴点の抽出

この節では 7.1.2 節の理論を元にして、実際に特徴点の抽出を行う。まず、図 7.3 のように最大特徴点数と浮動小数点数型座標の特徴点を定義する。

```
int i, key, count = 500;      // 特徴点の個数
char* status = new char[count];

CvPoint2D32f *feature_pre, *feature_now; // 浮動小数点数型座標の特徴点
feature_pre = new CvPoint2D32f[count];
feature_now = new CvPoint2D32f[count];
```

図 7.3: 特徴点の定義

また cvGoodFeaturesToTrack 関数により、画像中のコーナーやエッジなどの境界部分を検出し、特徴点として抽出する（図 7.4）。具体的には、第 1 引数 frame_pre_face の中から特徴点（コーナー）を検出し、第 4 引数に指定された feature_pre に結果を格納する。feature_pre は浮動小数点数型の 2 次元座標値を 500 点まで格納できるように定義した配列である。第 2 引数（img_tmp1）と第 3 引数（img_tmp2）は、一時的に使用される計算用配列である。第 5 引数の count には検出された特徴点の個数が格納される。第 6 引数にはコーナーのクオリティの許容最小値を「0.0＜第 6 引数＜1.0」の範囲で指定する。この値が小さいほど検出数が増加する。第 7 引数では、各コーナー間の距離の許容最小値を 0～30 程度の値で指定する。検出された特徴点同士がこの距離より近い場合、コーナー強度が弱い方が削除される。この値が小さいほど密に特徴点が抽出される。第 8 引数はマスク画像だが、使用しないため NULL を指定する[35]。

```
// pre_face からの特徴点の抽出
cvGoodFeaturesToTrack(frame_pre_face, img_tmp1, img_tmp2,
                                        feature_pre, &count, 0.001, 5, NULL);
```

図 7.4: 特徴点の抽出

特徴点抽出の様子を図 7.5 に示す。

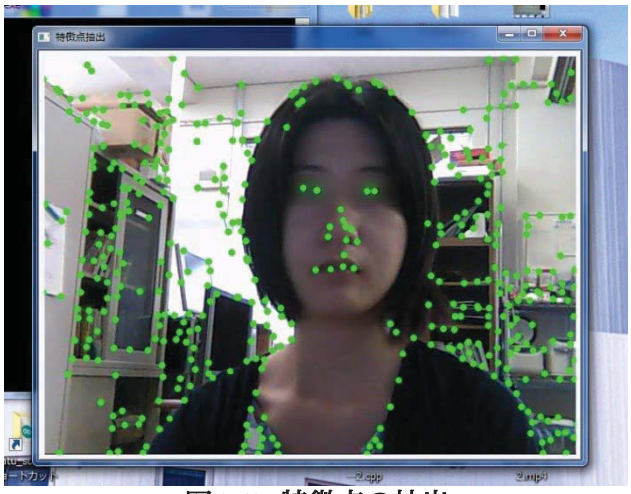


図 7.5: 特徴点の抽出

7.1.4 オプティカルフロー

オプティカルフローとは、動画像中の物体の動きをベクトルで表現する方法である。これを前節の特徴点抽出と組み合わせて用いる。すなわち、特徴点の座標の変化から、映像フレーム間における特徴点の移動を線分で結ぶとオプティカルフローが得る[35]。

7.1.3 節の方法で特徴点を抽出した後、図 7.6 のように、1 つ前のフレーム画像 frame_pre と現在のフレーム画像 frame_now の 2 フレーム間のオプティカルフローを Lucas-Kanade 法[39,40]を用いて検出する。pyramid_pre、pyramid_now はこのアルゴリズムで使用されるピラミッド画像である。Lucas-Kanade 法とピラミッド画像については、次節にて説明する。

```
// オプティカルフロー検出
cvCalcOpticalFlowPyrLK(frame_pre_face, frame_now, pyramid_pre, pyramid_now,
                                        feature_pre, feature_now, count, cvSize(10, 10),
                                                4, status, NULL, criteria, 0);
```

図 7.6: オプティカルフローの検出

また図 7.7 のように、frame_pre と frame_now の間にそれぞれ白い線分を描画する。そして、画像全体におけるオプティカルフローの平均を求める。

```
int ave_x = 0;
int ave_y = 0;
int ave_count = 0;

for(i = 0; i < count; i++){                    // オプティカルフロー描画
        cvLine(img_out, cvPointFrom32f(feature_pre[i]),
        cvPointFrom32f(feature_now[i]), CV_RGB (255, 255, 255), 1, CV_AA, 0);
```

```
double x_abs = fabs((double)(cvPointFrom32f(feature_now[i]).x -
                cvPointFrom32f(feature_pre[i]).x)); // ベクトル x 成分の絶対値
double y_abs = fabs((double)(cvPointFrom32f(feature_now[i]).y -
                cvPointFrom32f(feature_pre[i]).y)); // ベクトル y 成分の絶対値

            if(x_abs < 640/4 && y_abs < 480/4){//    「&&」は条件 A「かつ」B
                    ave_x += cvPointFrom32f(feature_now[i]).x –
                                cvPointFrom32f(feature_pre[i]).x;
                    ave_y += cvPointFrom32f(feature_now[i]).y –
                                    cvPointFrom32f(feature_pre[i]).y;
                    ave_count++;
             }
}

ave_x *= 10;
ave_y *= 10;

if(ave_count != 0){     //1 個でも点があったら
      ave_x /= ave_count;
      ave_y /= ave_count;
}
```

図 7.7: オプティカルフローの描画と平均

オプティカルフローの様子を図 7.8 に示す。白い線が特徴点の移動ベクトルを示し、赤い線は全てのベクトルを平均したベクトルである。

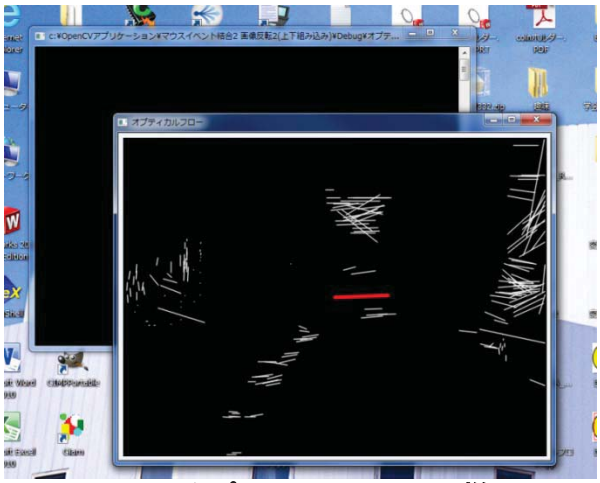


図 7.8: オプティカルフローの様子

7.1.5 Lucas-Kanade 法

Lucas-Kanade 法とは、着目する複数の点をそれぞれ囲む、小さなウィンドウを元にオプティカルフローを計算する手法である。ローカルな情報のみを使用するため、大まかな動きを求める際に有効である。

LK アルゴリズムは以下の 3 つの仮定を元に作成されている（図 7.9）。

1、明るさの不変性

オブジェクトの画像ピクセルは、フレームからフレームへ動くことがあっても外見上は変化しない。グレースケール画像では、フレーム間でトラッキングする前後において、ピクセルの明るさは変わらないことを意味する。

2、時間的な連続性

画像内の差分はフレームレートに対して、ゆっくり変化する。オブジェクトは連続する 2 フレーム間では、大きく動かない。

3、空間的な一様性

シーン内の隣接する点は同じ面に属し、似たような動きをする。そして画像平面上の近くの点に投影される。

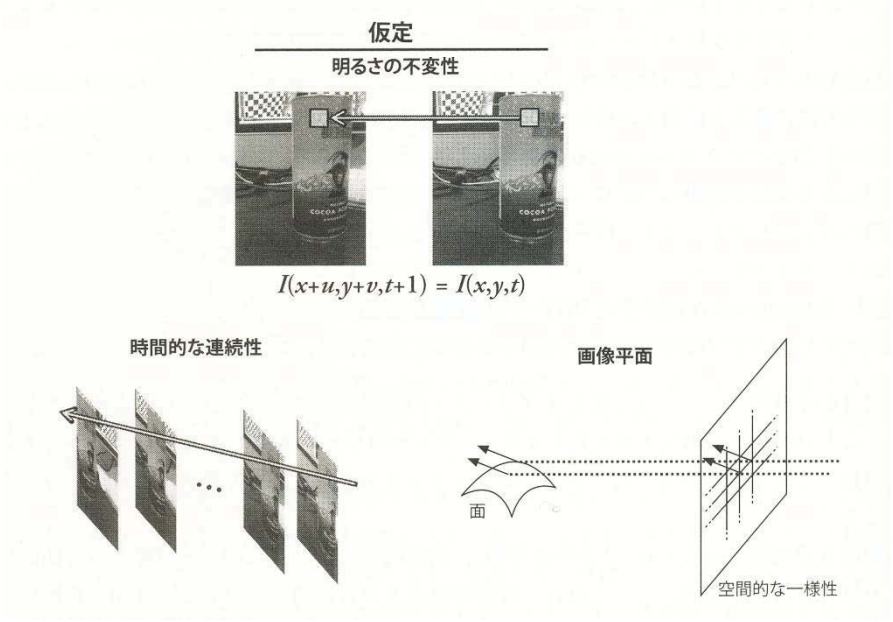


図 7.9: LK アルゴリズムの仮定[46]

この手法の欠点は、対象が大きな動きをしたときに、点が小さなウィンドウの外に出てしまい、このアルゴリズムでは点を発見できなくなることである。この問題を解決するために「ピラミッド型」LK アルゴリズム（LK：Lucas-Kanade）を用いる。これは、最もレベルの高い画像ピラミッド（大まかな画像）からトラッキングを始め、より低いレベル（細部が詳細な）へのトラッキングを行う。画像ピラミッドを用いることで、ローカルなウィンドウでも大きな動きを捉えることが可能になる。この手法では、図 7.10 のような画像ピラミッドを用いて、最初により大きな空間でトラッキングを行い、画像ピラミッドを元となる画像ピクセルに到達するまで降りていくことにより、モーションの速度の推定値を修正していくことが可能になる。従って、最初に最も上のレイヤでオプティカルフローを求め、求めた結果として得られるモーションの推定値を下のレイヤで出発点として用いることが推奨される。この方法によって、モーションの仮定が破られることを最小限にすることができる。また、より高速であり、かつより長いモーションをトラッキングすることが可能になる[46]。

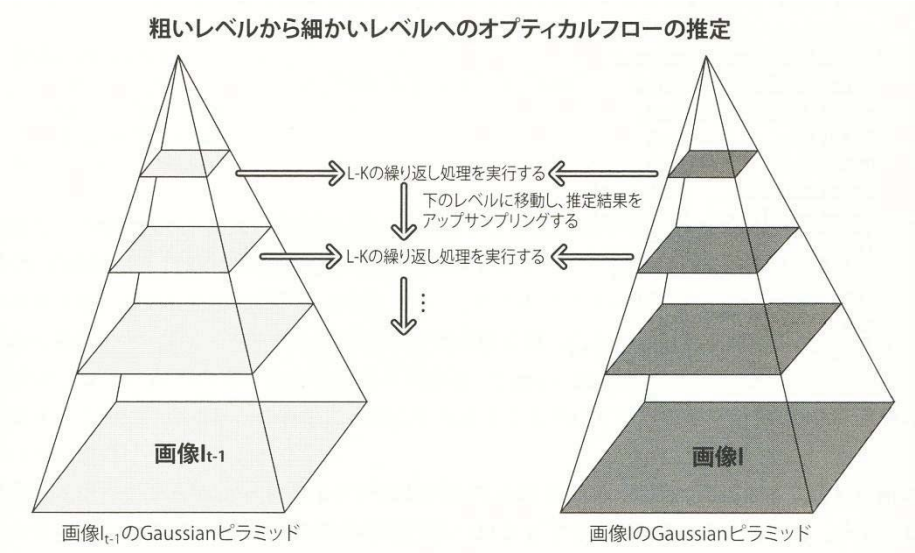


図 7.10: ピラミッド型 Lucas-Kanade 法のオプティカルフロー[46]

7.1.6 領域とオプティカルフローの組み合わせによるカーソル操作

7.1.4 節で求めた平均移動ベクトルを元に、マウスカーソルを移動する。そのために、図 7.11 のようにウィンドウを上下左右の 4 領域に分け、オプティカルフローの平均ベクトルが入った領域に従い、マウスカーソルの移動方向を決定する。

$y = -\frac{3}{4}x$

$y = \frac{3}{4}x$

図 7.11: 領域

Web カメラは本来の見たままを撮影するため、日常生活で鏡像を見慣れていると左右の移動が逆転している様に感じてしまう。よって動作確認をしやすくするため、図 7.12 ように cvFlip 関数を用いて画像の左右を反転する。また、反転後の 4 領域は下記の通りである。

$$\text{上} : y \geq \frac{3}{4}x,\ y > -\frac{3}{4}x \qquad \text{下} : y \leq \frac{3}{4}x,\ y < -\frac{3}{4}x$$

$$\text{左} : y \geq -\frac{3}{4}x,\ y < \frac{3}{4}x \qquad \text{右} : y \leq -\frac{3}{4}x,\ y > \frac{3}{4}x$$

```
cvFlip(img_out, img_out, 1);                    // 左右反転（鏡面モード）
cvFlip(frame, frame, 1);                        // 左右反転（鏡面モード）
```

図 7.12: 左右反転

図 7.13 では、右領域にオプティカルフローの平均があった場合にマウスカーソルを右に動かしている。

```
…
…
//向き C　右→ ピンク
//ピンク色  CV_RGB (255, 153, 160)
if(ave_y <= -3*ave_x /4 && ave_y > 3*ave_x /4){
    po.x += 30;
    SetCursorPos(po.x ,po.y);

    cvLine(img_out, cvPoint(320,240), cvPoint(320+ave_x,240+ave_y),
                CV_RGB (255, 153, 160), 4, CV_AA, 0); //オプティカルフロー  ピンク線

    }
count_click = 0;
}
```

図 7.13: オプティカルフロー平均が右領域の場合にカーソルを右に移動

7.1.7 実行結果と問題点

WindowsPC 上で動作確認を行ったところ、オプティカルフローの平均ベクトルが入った領域に従い、マウスカーソルを移動することができた。一方で、以下の 2 つのような問題が浮き彫りになった。

【問題 1】画面の端へのカーソル移動が難しい

ここで扱った手法ではマウスを動かし続けるために、同じ方向に顔を動かし続ける必要がある。しかし、首には可動域の限界があるため、同一方向へ動かし続けることはできない。パソコンの画面の端にカーソルを移動する際に途中で可動域の限界がきてしまうため、同じ方向へ動かし続けるには一度首を戻す必要がある。すると逆方向へ平均ベクトルが入ってしまい、マウスカーソルが逆方向に移動してしまう。

この問題を解決するためには、顔の移動ベクトルの他にも情報が必要である。さらに、人間の首の可動域も考慮する必要がある。首の可動域を表 7.1（20 代～80 代）、高齢者（50 代～80 代）の首の可動域を表 7.2 に示す[40,41]。

表 7.1: 首の可動域（2001～2002 年）[40,41]

部位	項目	データ数	平均値	Max	Min	5%tile	95%tile	標準偏差
頚部	左側屈	2832	39.74	81.00	13.00	25.00	55.00	9.15
頚部	右側屈	2826	−37.99	−6.00	−66.00	−53.00	−24.00	9.07

表 7.2:高齢者の首の可動域（2001～2002 年）[40,41]

部位	項目	データ数	平均値	Max	Min	5%tile	95%tile	標準偏差
頚部	左側屈	2166	38.22	69.00	13.00	24.00	54.00	8.80
頚部	右側屈	2166	−36.39	−6.00	−64.00	−52.00	−23.00	8.80

【問題 2】マウスカーソルの停止や移動方向の切り替えが難しい

ベクトルの反応が良すぎるため細かいマウスカーソル移動が難しい。またベクトルの停止範囲が少なく、移動や停止の範囲を使用者が判断しにくいこともマウスカーソルの停止や移動方向の切り替えが難しくしている。

この問題を解決するためには、それらの位置を視覚化する必要があると考えた。詳細は 7.4 節にて説明する。

7.2 顔の位置によるマウスカーソルの移動

前節で述べた【問題 1】を解決するために、顔の移動ベクトルの他に顔の位置情報を組み合わせることにした。この節では Haar-like 特徴分類法[42,43]を用いて顔と口の検出を行い、その情報を元にマウスカーソルの移動を行う。Haar-like 特徴では顔と口それぞれ検出できるので顔の位置によるマウス移動と口の位置によるマウス移動の両方を検討することにした。後に述べるように、最終的には安定性の高い顔の検出を用いることにした。

7.2.1 Haar-like 特徴分類法

Haar-like 特徴分類法は、人間の顔や眼、あるいはロゴマークのような形状の検出に適した分類方法である。このアルゴリズムでは、矩形（Haar 状）の分類器を検出対象画像と重ね合わせ、類似度を計算する。例えば図 7.14[35]のように、目の位置が黒、それ以外が白に近い明度をしている場合、分類機が一致する。

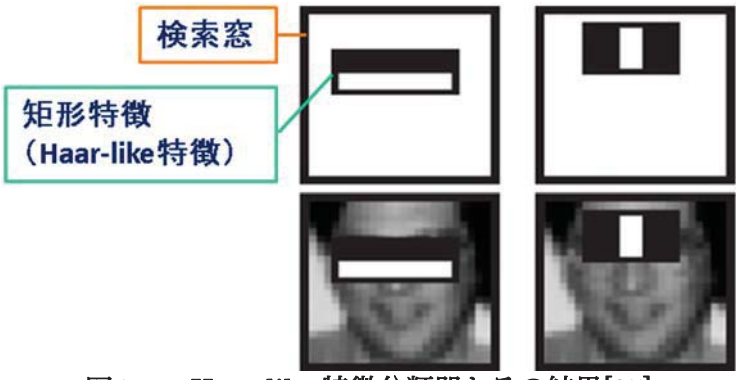


図 7.14: Haar-like 特徴分類器とその結果[35]

このような分類を、事前に大量のサンプル画像に対して行って学習させ、特徴ファイルを作成する。実行時には、これを外部ファイルとして読み込む。OpenCV には、機械学習を通じて作成された顔や眼などの特徴ファイルが用意されているため、それらを利用する[35]。

7.2.2 顔の検出

本節では OpenCV を用いてどのように顔を検出するか解説する。最初に図 7.15 のように、カスケード分類器を Haar-like 特徴分類構造体 CvHaarClassifierCascade で宣言する。また、メモリ領域を CvMemStorage で宣言する。

```
CvHaarClassifierCascade *cascade;        // カスケード分類器の宣言
CvMemStorage *storage;                   // メモリ領域の宣言
```

図 7.15: カスケード分類器とメモリ領域の宣言

次に図 7.16 のように、顔のカスケードファイル（特徴ファイル）である "haarcascade_frontalface_alt_tree.xml"を cascade に読み込み、Haar 検出で使用するメモリの確保と初期化を行う。なお、あらかじめ顔検出が使用するカスケードファイル"haarcascade_frontalface_alt_tree.xml"をプロジェクトフォルダにコピーしておく必要がある。

```
// カスケードファイルの読み込み
cascade_name = "haarcascade_frontalface_alt_tree.xml";
cascade = (CvHaarClassifierCascade *) cvLoad(cascade_name, 0, 0, 0);
storage = cvCreateMemStorage (0);            // メモリ領域の確保
cvClearMemStorage(storage);                  // メモリ領域の初期化
```

図 7.16: 顔のカスケードファイルの読み込み

図 7.17 のように顔の検出を行う。顔が検出されなかった場合 for 文は実行されない。cvGetSeqElem 関数は object から i 番目の検出結果の短形領域を取り出し、CvRect に短形の左上座標と高さを格納する。

また本研究ではマウスを使用する患者は 1 つのタブレットにつき 1 人である。そのためマウスカーソルの移動に使用する顔を複数検出しないように、画像の中で一番手前の顔、すなわち最も大きな顔を求める。

```
// Haar-Like 顔検出
objects = cvHaarDetectObjects(detect_frame, cascade, storage, 1.1, 3, 0, cvSize(0, 0));

int face_max_size = 0;
int face_max_index = -1;
CvRect *r_face_max=0;

for (int i = 0; i < (objects ? objects->total : 0); i++) {   // 検出数の繰り返し
// objects->total は検出物体数

    r = (CvRect *) cvGetSeqElem(objects, i);
      // 検出矩形領域の取得  r は顔の描画の起点座標


    //一番大きな顔を使用
    int length = r->width + r->height;
    if(length > face_max_size){
        face_max_size = length;
        r_face_max = r;
        face_max_index = i;
     }
}
```

図 7.17: 最も大きな顔の検出

検出した最も近い顔に cvRectangle 関数を用いて、四角い枠を赤色で描画する（図 7.18）。また図 7.19 のように cvShowImage 関数にて、検出結果を表示する。図 7.20 に顔検出の様子を示す。

```
cvRectangle(frame, cvPoint(r_face_max->x, r_face_max->y)  // 検出部に四角枠描画
                , cvPoint(r_face_max->x + r_face_max->width,
                        r_face_max->y + r_face_max->height), CV_RGB(255, 0, 0), 4); //赤
```

図 7.18: 検出結果に四角い枠を描画

```
cvShowImage ("顔検出", frame);    // 検出結果表示
```

図 7.19: 検出結果表示

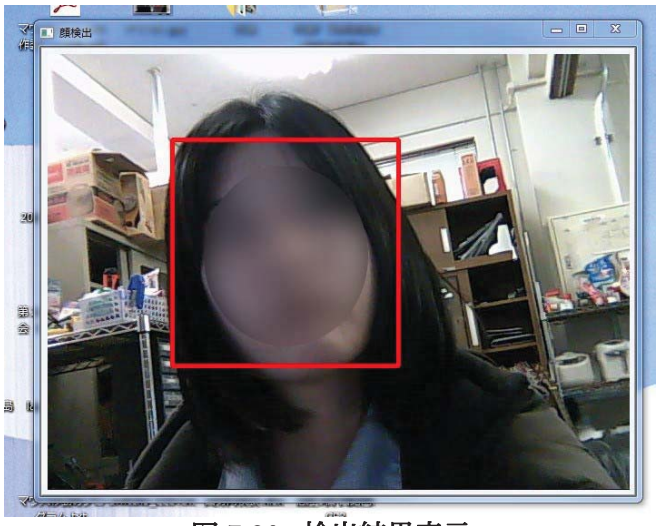


図 7.20: 検出結果表示

7.2.3 口の検出

口は眼と異なり、カーソル操作の際に無意識に動くことがない。また、顔の中の他の部位と比べて上下左右の 4 方向へ意識的に動かしやすい部位である。そして他の部位と色が異なるため、ユーザーが認識しやすい。よって、カーソルを動かす手がかりとして口は利用しやすいのではないかと考えた。

図 7.21 のように 7.2.2 節と同様にして、カスケード分類器とメモリ領域の宣言を行い、図 7.22 のように口のカスケードファイルである"haarcascade_mcs_mouth.xml"を cascade2 に読み込み、メモリの確保と初期化を行う。顔と同様に、口検出が使用するカスケードファイルをあらかじめプロジェクトフォルダにコピーしておく必要があることに注意する。

```
CvHaarClassifierCascade *cascade2;        // カスケード分類器の宣言
CvMemStorage *storage2;                   // メモリ領域の宣言
```

図 7.21: カスケード分類器とメモリ領域の宣言

```
// カスケードファイルの読み込み
cascade_name = "haarcascade_mcs_mouth.xml";
cascade2 = (CvHaarClassifierCascade *) cvLoad(cascade_name, 0, 0, 0);
storage2 = cvCreateMemStorage (0);          // メモリ領域の確保
cvClearMemStorage(storage2);                // メモリ領域の初期化
```

図 7.22: 口のカスケードファイルの読み込み

同様に口の検出を行ったところ、多くの誤検出が発生した（図 7.23）。誤検出を低減するために、図 7.24、図 7.25 のように最も大きな顔の下部を cvSetImageROI 関数で切り出した。

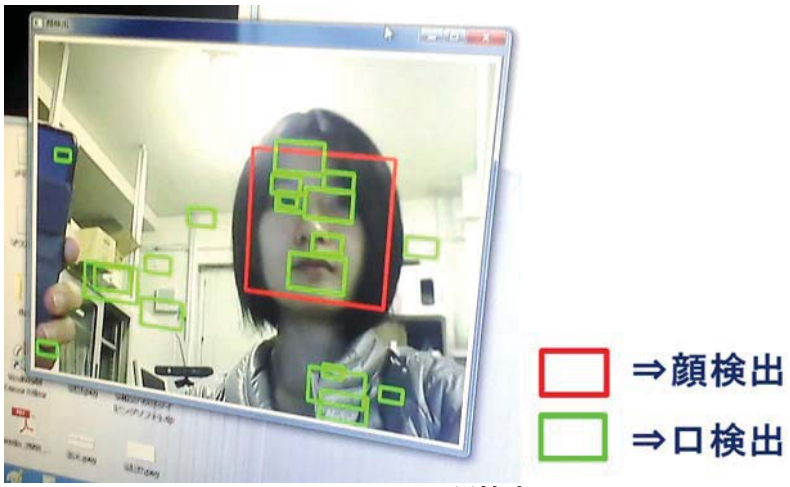


図 7.23: 口の誤検出

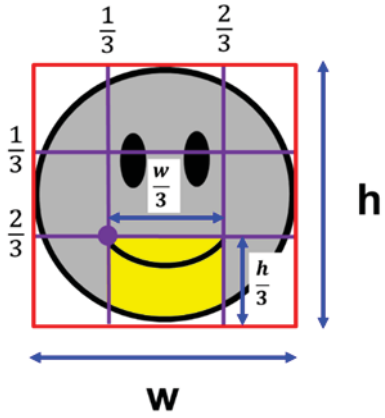


図 7.24: 切り出す領域

```
// この時点で、最大の顔のオブジェクトの矩形領域は r_face_max
// 顔の下半分の領域
    lower_half = cvRect(r_face_max->x + r_face_max->width / 3, r_face_max->y
                + 2*r_face_max->height / 3, //原点から 1/3 幅 x,2/3 の高さ y をずらす ①
                        r_face_max->width/3, r_face_max->height / 3); //縮尺
```

```
cvResetImageROI(detect_frame);
cvSetImageROI(detect_frame, lower_half);
```

図 7.25: 顔の下半分の領域

図 7.26 のように切り出した顔の下部の中で口の検出を行い、検出部に緑色の四角枠を描画した。顔が検出されない場合は、口の検出は行われない。図 7.27 に、誤検出低減後の様子を示す。

```
// Haar-Like 口検出
objects2 = cvHaarDetectObjects(detect_frame, cascade2, storage2,
1.1, 3, 0, cvSize(0, 0));

for (int i = 0; i < (objects2 ? objects2->total : 0); i++) {          // 検出数の繰り返し
        r = (CvRect *) cvGetSeqElem(objects2, i);                    // 検出矩形領域の取得
        // 検出部に四角枠描画
        cvRectangle(frame, cvPoint( lower_half.x + r->x, lower_half.y + r->y)
          , cvPoint(lower_half.x +r->x + r->width, lower_half.y +r->y + r->height),
                                                                CV_RGB(0, 255, 0), 4); //緑
}
```

図 7.26: 口の検出と検出結果の描画

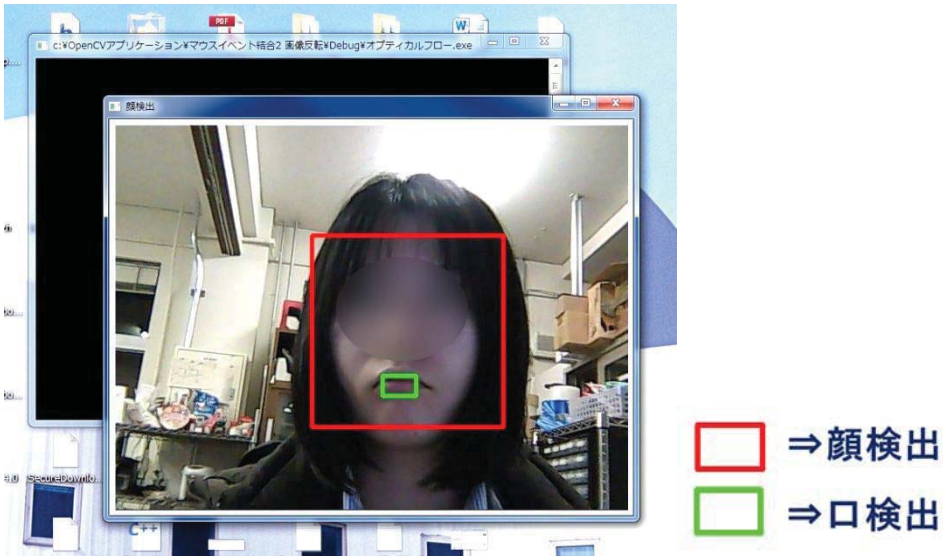


図 7.27: 口の誤検出の低減

7.2.4 口の位置によるマウスカーソルの移動

前節で検出したウィンドウ上の口の位置を用いてカーソル移動を試みた。図 7.28 では口の位置の座標を定義し、口が左の領域内にあるときにカーソルが左向きに移動する。

```
//口の中心座標の定義
int mouth_x = ( (lower_half.x +r->x + r->width)/2);
int mouth_y = ((lower_half.y +r->y + r->height)/2);

//①位置　左
if(380<mouth_x && mouth_x<640 && 0<mouth_y && mouth_y<480){
   po.x -= 30;
   SetCursorPos(po.x ,po.y);
}
```

図 7.28: 口による左向きのマウスカーソル移動

プログラムを実行したところ、検出した口が左移動の領域内にあるときにカーソルが左向きに移動した。しかし顔が正面を向いている場合でも、顔の角度によっては口を検出しないことがあり、その場合はカーソルの移動が行えなかった。これは、顔検出に比べて口検出の条件が厳しいためである。また顔の下部で口が複数検出されることや、口の検出枠の大きさが安定しないこと多々あり、口の中心座標が揺れてしまう問題がある。よってマウスカーソルの移動に使用する位置情報には、検出が確実でありかつ信頼性の高い情報である顔の位置情報を用いる方がよいと考えた。

7.2.5 顔の位置によるマウスカーソルの移動

7.2.4 節で挙げた問題により、マウスカーソルの移動に使用する位置情報を、口検出でなく 7.2.2 節で扱った顔検出に決定する。

ただし、口は顔の他の部位と色が異なるため、ユーザーが認識しやすく、顔の中の相対的な位置の個体差が少ないことから、口の位置にカーソル移動の目安があるとわかりやすいと考えた。そこで、顔検出後の赤い四角形の中の図 7.29 の位置をカーソル移動の目安とし、図 7.30 のように座標を定義する。図 7.31 のように、図 7.29 の位置に赤い丸を描画した。

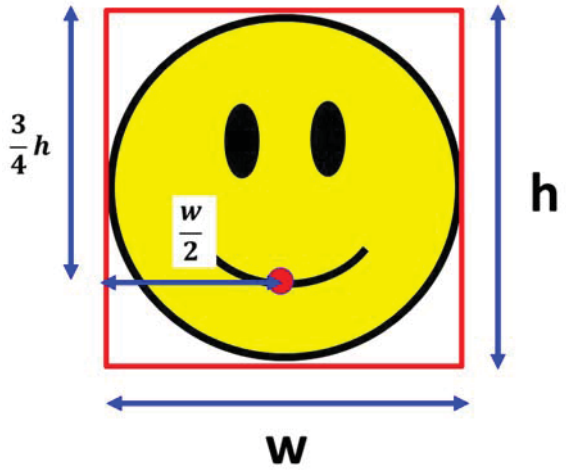


図 7.29: カーソル移動の目安となる顔の赤い丸の位置

```
// 顔の●の定義
int mouth_x = (r_face_max->x + r_face_max->width/2);
int mouth_y = (r_face_max->y + 3 *(r_face_max->height)/4);
```

図 7.30: カーソルに使用する位置情報の変更

```
//顔の(x,y)=(1/2, 3/4)に赤色の●を描画
cvCircle(frame /* 描画するフレームの指定 */,
        cvPoint(r_face_max->x + r_face_max->width/2, r_face_max->y + 3
        *(r_face_max->height)/4) /* 円の中心（顔の中心）座標 */,
    1/* 円の大きさ */, CV_RGB(255, 40, 0) /* 色 */, 16 /* 線の太さ */, CV_AA, 0);
```

図 7.31: 赤色の丸を描画

顔認識に用いたウィンドウを図 7.32 のように 8 つの領域に分け、赤色の丸がある領域に応じてマウス操作を行う。カーソル移動の単発動作（図中「左」および「右」）では、左右のカーソル移動における微調整を行う。例えば、赤色の丸の位置が「右」領域内にあり、かつオプティカルフローで検出した物体が右に動いている間、カーソルを右へ少し動かす。カーソル移動の連続動作（図中「左連続」および「右連続」）では、赤色の丸が領域内にある間、その領域にて指定されている方向へカーソルを移動させる（図 7.33）。また、クリックの領域内（図中「C」）に赤色の丸が一定時間入っている場合にクリックを行う。詳細は 7.3 節で述べる。

図 7.32: 操作盤の詳細

```
//⑥位置　右→
else if (r1x<mouth_x && mouth_x<x1 && y1<mouth_y && mouth_y<y3){

            //向きC　右→ピンク
            //ピンク色CV_RGB（255，153，160）
            if(ave_y <= -3*ave_x /4 && ave_y > 3*ave_x /4){
               po.x += 30;
               SetCursorPos(po.x ,po.y);

            //右矢印→（右向き三角）をピンク色で描画
            //ピンク色CV_RGB (255, 153, 160)
              cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
              cvPoint(32 + genten_idou_x ,80 + genten_idou_y), CV_RGB(255, 153, 160) ,
              12, CV_AA, 0); //
              cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
              cvPoint(32 + genten_idou_x ,90+ genten_idou_y), CV_RGB (255, 153, 160) ,
              12, CV_AA, 0); //｜
              cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
              cvPoint(32+ genten_idou_x ,90 + genten_idou_y), CV_RGB (255, 153, 160) ,
              12, CV_AA, 0); //／
            }
            count_click = 0;
}


//⑤位置　右→→
else if(x0<mouth_x && mouth_x<r1x && y1<mouth_y && mouth_y<y3){

            po.x += 30;
            SetCursorPos(po.x ,po.y);

            //右矢印｜→（右向き三角と線）をピンク色で描画
            //ピンク色CV_RGB (255, 153, 160)
            //三角
              cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
              cvPoint(32 + genten_idou_x ,80 + genten_idou_y), CV_RGB(255, 153, 160) ,
              12, CV_AA, 0); //
              cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
              cvPoint(32 + genten_idou_x ,90+ genten_idou_y), CV_RGB (255, 153, 160) ,
              12, CV_AA, 0); //｜
               cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
              cvPoint(32+ genten_idou_x ,90 + genten_idou_y), CV_RGB (255, 153, 160) ,
              12, CV_AA, 0); //／

               count_click = 0;
}
```

図 7.33: マウスカーソルの移動

7.3 顔の位置によるマウスクリック

図 7.34 のようにウィンドウ上の「C」の領域に、一定時間以上 7.3.5 節で描画した赤い丸が入っているときに、mouse_event 関数のパラメータの 1 つである MOUSEEVENTF_LEFTDOWN にて、マウスの左ボタンを押す。同様に MOUSEEVENTF_LEFTUP にて左ボタンを離すことで、クリックを行う。これをもう一度繰り返し、ダブルクリックを行う。最後に時間のカウントをリセットする。

```
…
        //クリック範囲
        if(x2<mouth_x && mouth_x<x3 && y1<mouth_y && mouth_y<y_c1){

        //10 カウント
        if (count_click < 10) {
                count_click = count_click+1; //count_click を 1 増やす
        }
        else{ // count_click が 10 回以上なら以下を処理
                //マウスクリック(2 回→ダブルクリック)
                mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
                mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
                mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
                mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);

                count_click = 0;
        }
```

図 7.34: 顔の位置によるマウスクリック

7.4 操作盤の作成

7.4.1 操作盤の作成

インターフェースの操作性の良さは、患者が継続的に利用し続けるための重要な条件の 1 つである。また 7.1.7 節の【問題 2】を解決するためにも、操作性が良いインターフェースを作成する必要がある。操作しやすいインターフェースとして以下の 3 つの条件を満たす必要があると考えた。

【条件 1】ひと目で操作の内容がわかる（＝説明がいらない）

患者がひと目で操作の内容を理解するために、カーソル動作の視覚化を行う。動作の内容をアイコン化し、アイコンの色を変化させる動作の変化を示すことで、直感的な操作を可能にする。アイコンに使用する図形は、一般的かつ簡易的なものが望ましい。

そこで、テレビリモコンを参考にアイコンを作成した。また、動作の変化をより伝えやすくするために、色覚の特性に配慮する必要がある。そのため、動作の変化に使用する色にはカラーユニバーサルデザイン[45]を用いる。色覚異常とカラーユニバーサルデザインについては次節にて述べる。図 7.35 のように動作の内容をアイコン化し、操作を行っているアイコンの色が変化することで、現在どんな操作をしているか一目瞭然である。また、カラーユニバーサルデザインを用いたことで、図 7.36 のように色覚の多様性に対応した。

【条件 2】操作の邪魔にならない

インターフェースによってマウス操作自体を阻害しないためには、操作盤が一箇所へまとまっており、動作（カーソル操作）の邪魔にならない配置とサイズであることが望ましい。一方で、操作箇所は大きい方が操作はしやすい。

操作の内容をシンプルにすることで、操作箇所の一箇所あたりのサイズを大きくする。また、邪魔にならない操作盤の配置は人によって異なるため、操作盤は任意の位置に動かせる方がよい。

【条件 3】人の動きと操作がかみあっている

7.1.7 節の【問題 1】で挙げたように、人間の身体の可動域には限界と個体差がある。連続でカーソル移動を行う領域をつくることにより、可動域内で操作を行えるようにする。また、同じ方向へ首を動かし続ける際に、一度首を戻すなど、ある動作を行う際に無意識に行いやすい行動がある。それらに反応して誤動作を起こさないために、左右方向へ移動方向のベクトルと、顔の位置が一致する場合のみカーソル移動を行う領域を作成する。

また使用者はマウスを常に動かしているわけではない。あえて何も動作しない領域を作ることでカーソルの停止を容易にさせ、次の動作を行うまでの余裕を作ることができる。図 7.37 に操作盤の詳細について示す。

図 7.35: カーソル移動時のアイコンの色の変化

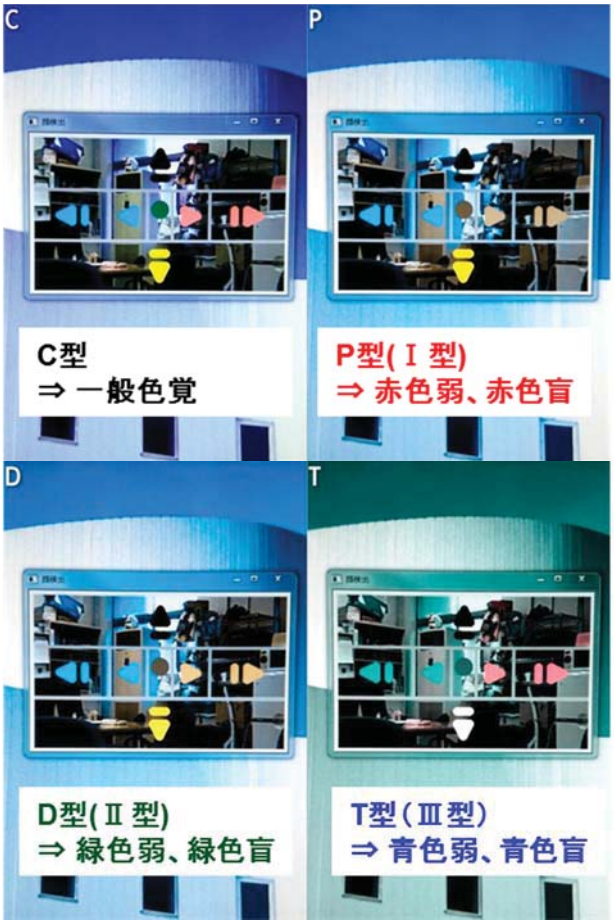


図 7.36: 色覚の多様性による操作盤上のアイコンの見え方

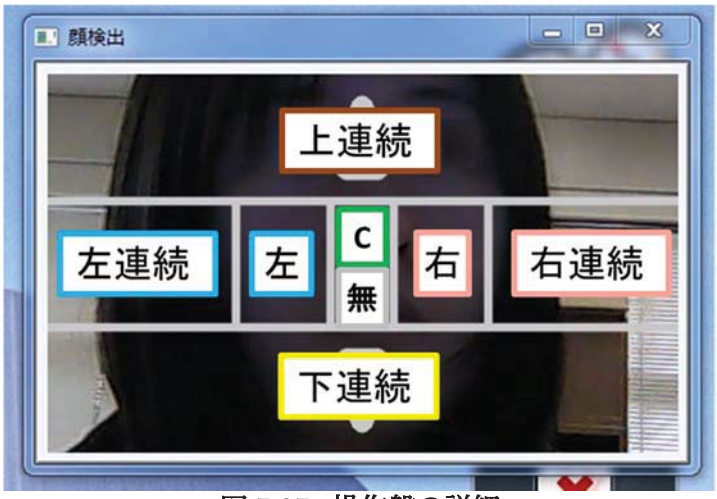


図 7.37: 操作盤の詳細

以上の条件を元にして、図 7.38 のように 640×480 ピクセルのウィンドウの中心部を 400×240 ピクセル切り出して、操作盤を作成した。中心部を切り出すことでウィンドウから顔がはみ出していても検出することができる。

```
cvSetImageROI(frame,cvRect(x0,y0,x5-genten_idou_x,y3-genten_idou_y));//ROI指定(切り出し)
cvResetImageROI(frame);//これをしないとROIのみ表示される
```

図 7.38: ウィンドウの切り出し　：

7.4.2 色覚の多様性

本節では、色覚の多様性について解説する。眼には、暗いところだけで使われる杆体と、明るいところで使われる錐体の、2 種類の視細胞がある。錐体には、長い波長を感じる L 錐体、中ぐらいの波長の光を感じる M 錐体、短い波長を感じる S 錐体の 3 種類があり、これらの反応の差を計算して人は色を知覚する（図 7.39、図 7.40）。色の見え方には様々な個人差があるが、中でも錐体の遺伝子や網膜の疾患、白内障などの要因は、色の見え方に大きく影響する。また、日本では男性の約 5%、女性の 0.2%（男性の 20 人に 1 人、女性の 500 人に 1 人）が錐体による色覚異常を持っており、数百万人いる[44]。

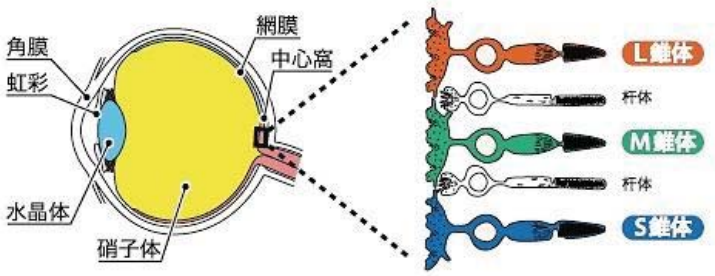


図 7.39: 眼と錐体[44]

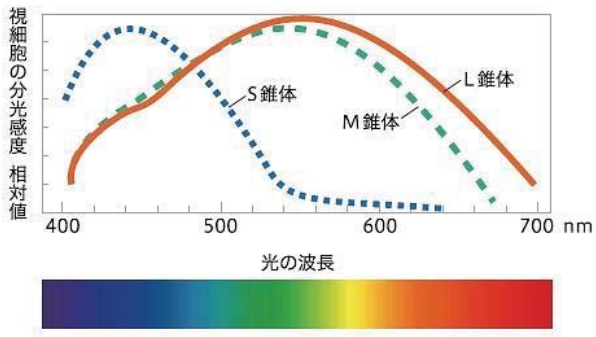


図 7.40: 錐体の分光感度[44]

7.4.3 カラーユニバーサルデザイン

7.4.2 節の理由から、一部の色の組み合わせを区別しにくく不便を感じる人がいる。カラーユニバーサルデザインとは、どのような色覚の人にも比較的見分けやすい配色にすることで、全ての人に情報が正確に伝わるよう配慮されたデザインのことである。

7.5 首振り動作による顔と口の検出量の比較

7.2 節で顔検出によりカーソル移動を行うことに決定したが、顔と口の検出精度をより詳細に比較することにした。この節では 7.4 節で作成した操作盤に顔検出の赤丸と、口検出の緑色の四角枠を表示し、図 7.41 の各動作枠内に顔検出の赤丸が入った際の、顔と口の検出量の違いについて比較を行う。実験の対象者は 20 代の男性 5 名である。

図 7.41: 操作盤の詳細

7.5.1 実験方法

各動作の領域に顔検出の赤丸が入った際の、顔と口の検出量の違いについて比較を行う。動作領域については図 7.41 を参照する。

①「無領域（何も動作のしない領域）」に赤丸を移動し、顔と口の検出を確認する。
②「無領域」から「右領域」に赤丸を移動し、顔と口の検出を確認する。
③「右領域」から「右連続領域」に移動し、顔と口の検出を確認する。
④「無領域」に戻る。
⑤「無領域」から「左領域」に赤丸を移動し、顔と口の検出を確認する。
⑥「左領域」から「左連続領域」に移動し、顔と口の検出を確認する。
⑦「無領域」に戻る。
⑧「無領域」から「C 領域(マウスクリックの領域)」に赤丸を移動し、顔と口の検出を確認する。
⑨「C 領域」から「上連続領域」に赤丸を移動し、顔と口の検出を確認する。
⑩「無領域」に戻る。
⑪「無領域」から「下連続領域」に赤丸を移動し、顔と口の検出を確認する。
⑫「無領域」に戻る。

① ⑫までを 1 セットとし、合計 50 セット繰り返す。そして各領域での顔と口の検出数を数え、検出量の比較を行う。実験の様子を図 7.42 に示す。また、使用した顔検出と口検出のチェックシートを付録に記す。

図 7.42: 実験の様子

7.5.2 実験結果

顔検出と口検出の検出数の平均を表 7.3、被験者ごとの検出数のグラフを図 7.43～図 7.47 に示す。同時に 2 個以上検出した場合と、検出したものの測定領域と別の場所に検出枠が表示された場合を誤検出とする。また、検出枠が全く表示されない場合は未検出とする。

実験から顔検出は口検出に比べて検出数が多く、口は検出されてはいるものの同時に 2 個以上検出される誤検出が多いことが分かった。また、口検出は検出の途中で一瞬消えることがあり、検出が不安定であった。以上により、操作には安定性の高い顔検出を引き続き用いることにした。

表 7.3 より、操作盤の領域別にみると右連続領域、左連続領域、上連続領域の検出数が低く、また被験者も上記 3 領域へ赤丸を入れることが難しいと答えた。一方で、他の連続領域が他の領域との境界付近で顔を検出しているのに対して、下連続領域には余裕があり、同領域内の真ん中からウィンドウの下端付近で顔を検出していた。また、首振りは「上下に比べて左右の方が動かしやすい」との意見があった。実験時間は 1 人あたり約 50 分から 60 分であったが、被験者 5 名中 4 名は身体への負荷について「全く問題ない」と答えた。しかし、1 名は「首の付け根が張るような感じがする」と答えた。原因を被験者に聞いてみたところ「操作する領域が広いと、首を伸ばさなくてはいけない気分になる」と答えており、被験者が無理をしていたことが分かった。そこで使用者が無理をせず使うことができ、かつ顔検出が安定している領域で操作することができるよう操作盤を改善することにした。詳細は 7.6 節で述べる。

表 7.3: 顔と口の検出数（平均）

	検出(回)		誤検出(回)		未検出(回)	
領域	顔検出	口検出	顔検出	口検出	顔検出	口検出
無領域	50	16.8	0	32.4	0	0.8
右領域	49.6	19.4	0	28.4	0.4	2.2
右連続領域	34.8	10.8	10	17	5.2	22.2
左領域	44.8	9.4	4.4	16.4	0.8	24.2
左連続領域	12	1.8	16	8.6	22	39.6
C 領域	49.6	22.4	0.4	22	0	5.6
上連続領域	35.4	14.6	10.6	11.4	4	24
下連続領域	47.6	9	2.2	40	0.2	1

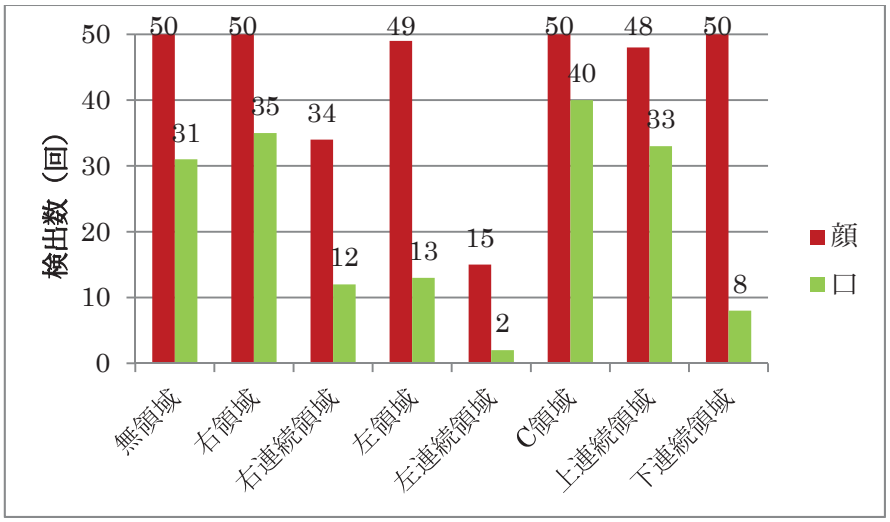


図 7.43: 被験者 1 の検出回数

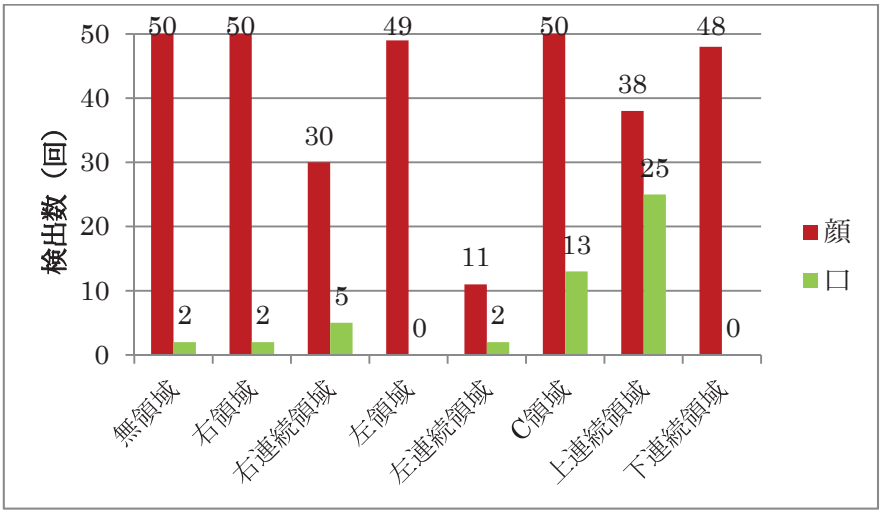


図 7.44: 被験者 2 の検出回数

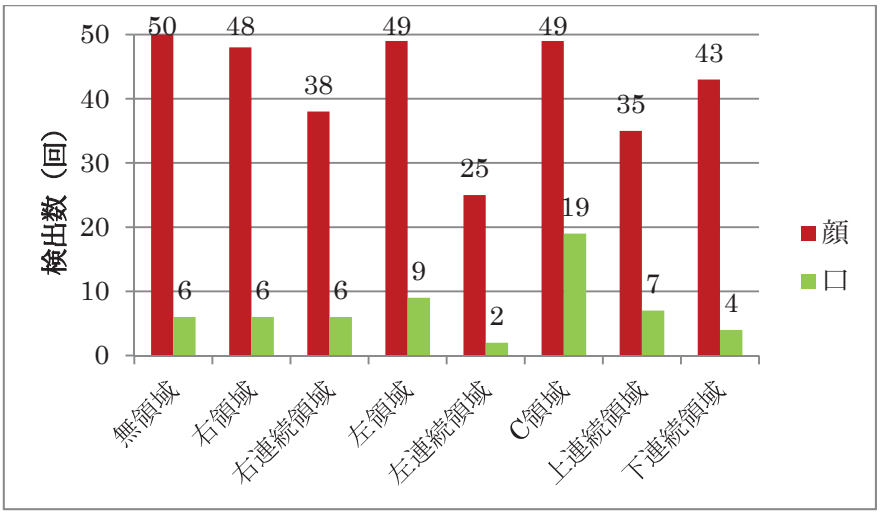


図 7.45: 被験者 3 の検出回数

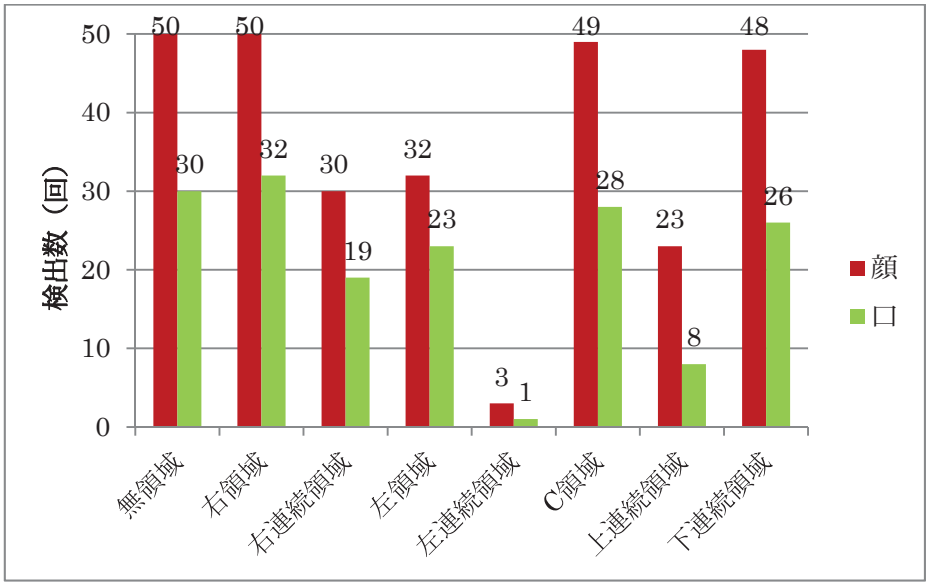


図 7.46: 被験者 4 の検出回数

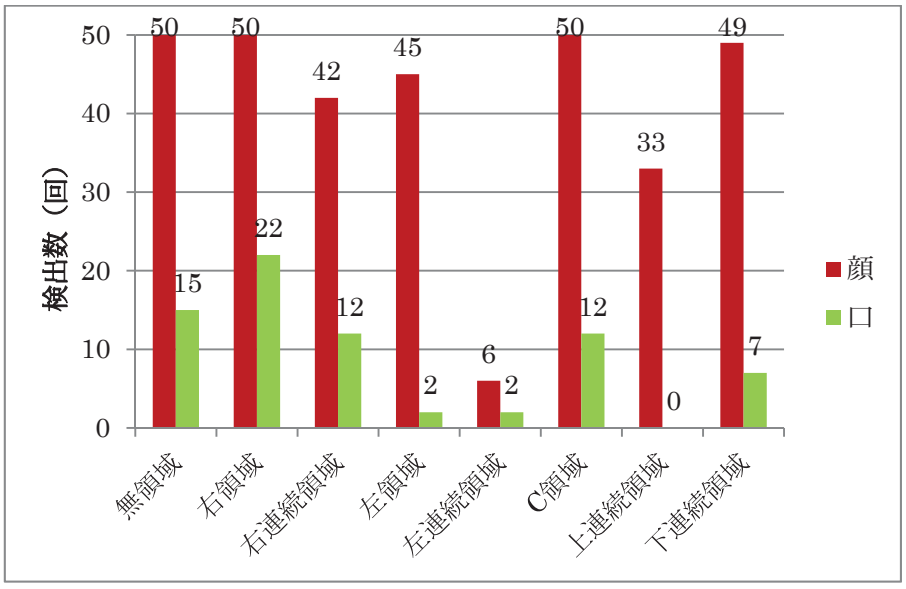


図 7.47: 被験者 5 の検出回数

7.6 操作盤の改善

7.5 節の結果から、操作盤を小さく修正することにした。単に縮小するのではなく実験中の被験者の顔の動きを踏まえて、次の通りに修正する。上下の連続領域の範囲を縮小し、左連続領域と左領域の範囲内に組み込んだ。右連続領域と右領域も左と同様し、C 領域と無領域を広げた。詳細は図 7.48 の通りである。図 7.49 のように最終的に操作盤の大きさは 400×240 ピクセルから 160×160 ピクセルまで小さくしたものの、一つ一つの操作領域の大きさを必要十分に残したことにより、操作がしやすいデザインになっている。

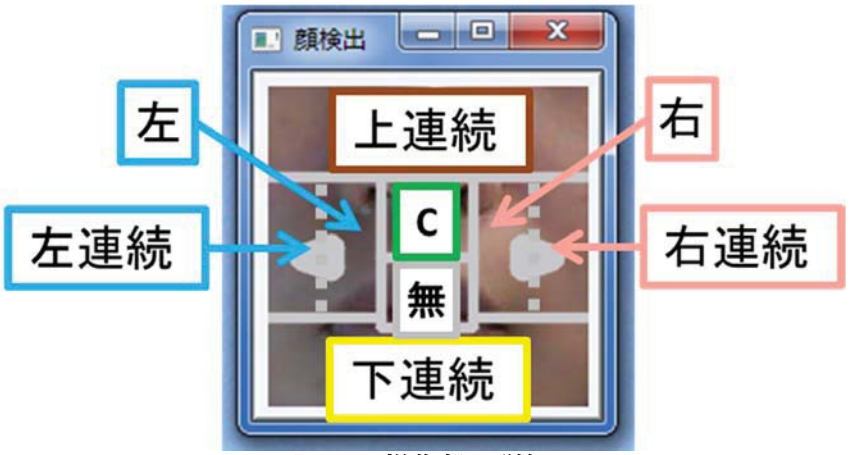


図 7.48: 操作盤の詳細 2

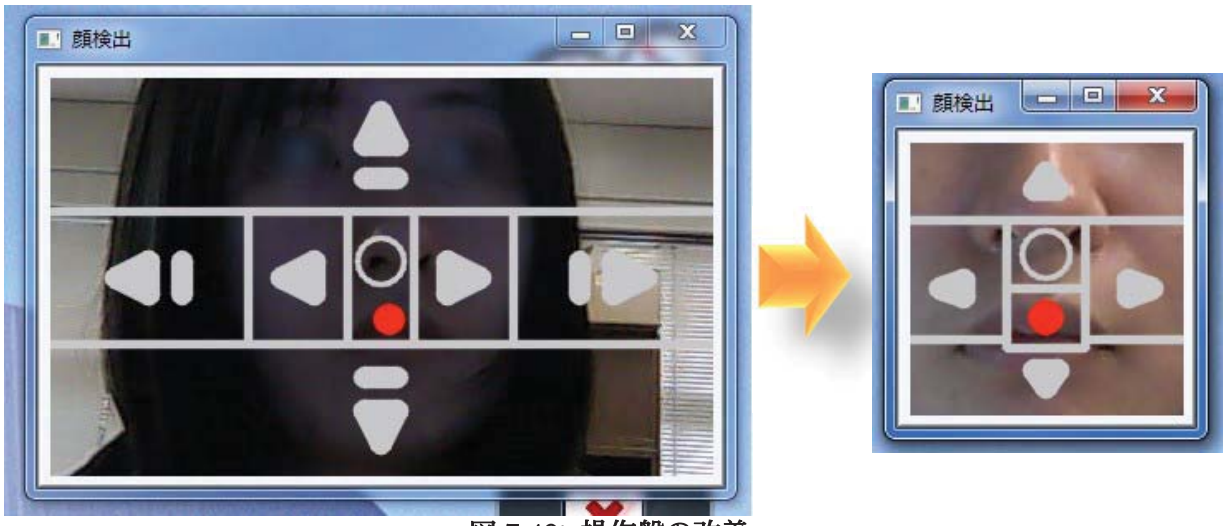


図 7.49: 操作盤の改善

7.7 Windows タブレットへの統合

前節まででデスクトップ PC 上にて動作確認を終えたため、本節では Windows タブレットへの統合を行う。デスクトップ PC では映像の取得に USB カメラを使用していたが、Windows タブレットではインカメラを使用する。そのためには、cvCaptureFromCAM 関数のカメラを 0 から 1 へ変更する。またタブレットによってカメラの取得画像のサイズが異なるため、cvSetCaptureProperty 関数を用いてカメラから取得した画像の高さと幅を指定する（図 7.50）。また、図 7.51 に windows タブレット上での実行結果を示す。顔の移動と位置によるマウス操作のプログラムの全文を付録に記す。

```
//カメラの選択
//Windows タブレット　0=アウトカメラ(rear camera)、1=インカメラ(front camera)、
2=USB カメラ（外付け）
//PC（PC にカメラがついてない場合）　0=USB カメラ（外付け）

src = cvCaptureFromCAM(1);　　// 映像取得（カメラ映像）

//カメラから取得した画像のサイズ指定
cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_WIDTH, 640);
cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_HEIGHT, 480);
```

図 7.50 ：使用するカメラの選択と画像サイズの指定

図 7.51：windows タブレット上での実行結果

第 8 章 評価

この章では第 7 章の画像処理によるマウスカーソルの操作による評価と考察を行う。評価するにあたって 3.2.1 節と 7.4.1 節の条件、そして「ユニバーサルデザイン 7 原則」を満たしているか、否かを 5 段階で評価する。

8.1 評価項目

評価項目は以下の通りである。

① ヒューマンインタフェースの条件（3.2.1 節）

【条件 1】脊髄損傷患者による操作を検知できる
【条件 2】安全である
【条件 3】衛生が良い（＝清潔を保てる）
【条件 4】直感的に操作ができる
【条件 5】コストが安い（＝病院などに提供できる値段である）
【条件 6】患者のプライバシーが守られる（＝操作内容を外部へ漏らさない）

② 操作盤の条件（7.4.1 節）

【条件 1】ひと目で操作の内容がわかる（＝説明がいらない）
【条件 2】操作の邪魔にならない
【条件 3】人の動きと操作がかみあっている

③ ユニバーサルデザイン 7 原則[49]

【条件 1】公平性
【条件 2】柔軟性
【条件 3】直観性（単純性）
【条件 4】マルチモーダル性
【条件 5】対エラー性
【条件 6】低労力性
【条件 7】大きさ・空間の確保

①の条件の内、条件 5 は今回の方式で評価を行えないため、評価項目から除外する。また、③の条件の内、①②と重複する条件 3、5 を外し、今回の方式で評価できない条件 4、7 は 8.2.2 節にて述べる。また、ユニバーサルデザイン 7 原則については、次節にて述べる。

8.1.1 ユニバーサルデザイン 7 原則

ユニバーサルデザインでは、障害を持つ人や高齢者に対応した仕様を基準とするが、健常者や若年層までできるだけ多くのユーザーが使えるよう考慮する必要がある。ノースキャロライナ州

立大学（米）の、ロン・メイスらが中心となり提唱したユニバーサルデザイン 7 原則[49]は以下の通りである。

【条件 1】公平性
身体条件の差、年齢差、性差、国籍などの違いに関わらず、あらゆるユーザーが公平に使うことができ、不満を抱かせない。

【条件 2】柔軟性
使い方に関する制約が少ない、あらゆるユーザーが自分のやり方、自分のペースで使用できる。

【条件 3】直観性（単純性）
知識や経験、文化の違いに関わらず、どのユーザーにも使い方がすぐにわかる。

【条件 4】マルチモーダル性
視覚、聴覚、皮膚感覚など、ある 1 つの手段だけでなく、色々なモダリティで使える。

【条件 5】対エラー性（安全性）
色々なユーザーが使うことを前提に、起こりうる様々なエラーに対して、予防的対策および対蹠的対策を備え、頑健にしておく。

【条件 6】低労力性
少しの力で使うことができ、疲労も少ない。

【条件 7】大きさ・空間の確保
ユーザーの身長やバリエーション、車椅子利用の有無、使う姿勢などを考慮し、十分な大きさやスペースを持つ。

8.2 評価実験

8.2.1 実験方法

被験者に、使用方法の説明はせずに、1 分間自由にインターフェースを使用してもらう。その後、図 8.1 のように、3 つのアイコンを画面に配置し、息圧と画像処理それぞれのインターフェースで下記の作業を行う。同じ作業を 3 回繰り返し、作業をはじめてから終わるまでの時間を比較する。実験の対象者は 20 代の男性 5 名である。使用したアンケートを付録に記す。

①画面の中心 A に置いたカーソルをから右下のアイコン B に移動する。
②アイコン B から左上のアイコン C にカーソルを移動する。
③アイコン C からアイコン A にカーソルを移動する。
④アイコン A をクリックし、ファイルを開く。

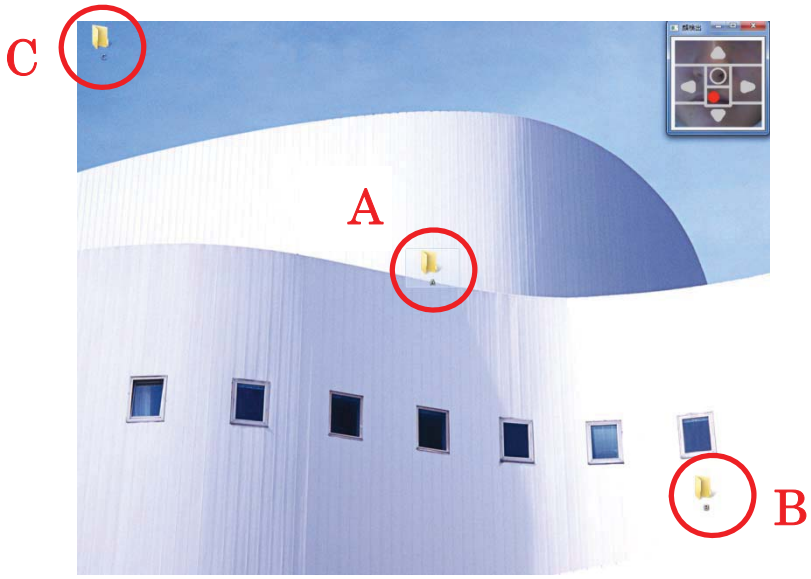


図 8.1: アイコンの配置

8.2.2 実験結果

実験結果を表 8.1、アンケートの結果を表 8.2～8.4、図 8.2～8.4 に示す。表 8.1 より、長くとも 100 秒程度で作業を行うことができた。また、被験者の操作で共通して時間がかかった動作はアイコン付近におけるマウスカーソルの左右方向への微調整である。被験者からは「左右の移動が速い」「左右の動きは普段は丁度いいけど、アイコンの近くだと早い」「左右の反応が少し悪い」との意見が出た。上下方向と左右方向のマウスカーソルの移動速度が同じにもかかわらず、左右方向の移動速度のみ意見が集中したのは、左右方向の移動の方が上下方向より繊細な調整が必要であるからと考えた。次節にて、左右方向の移動速度の修正を行う。

また表 8.2、図 8.2 より、ヒューマンインタフェースの条件のうち条件 1～4 は 5 段階評価で平均が 4 以上であった。「【条件 6】患者のプライバシーが守られる」の評価の平均が 3.4 と低い理

由として「首を動かすことでマウスの操作をしているのがわかる気がする」との意見があり、操作内容以前に作業自体を他者に認知されるのが気になる人も居ることがわかった。

表 8.3、図 8.3 より、操作盤の条件のうちの条件 1、3 の評価は 5 段階評価で平均が 4 以上であった。「【条件 2】操作の邪魔にならない」の評価の平均が 2.8 と低い理由として「ウィンドウと操作盤被った場合にその部分が見えない気がする」との意見があった。

表 8.4、図 8.4 より、ユニバーサルデザイン 7 原則のうち条件 2、6 は 5 段階評価で平均が 4 以上であった。「【条件 1】公平性」の評価の平均は 3.2 であった。条件 4、7 については、今回の方式で評価できないため以下で考察を行う。

【条件 4】マルチモーダル性

本研究のメインとなるインターフェースは画像処理であるが、万が一、一定方向のみ首が動かしづらい場合に、その向きのカーソル操作を画像処理の代わりに第 6 章の息圧のインターフェースで代用することができる。

【条件 7】大きさ・空間の確保

図 8.5 の通り、脊髄損傷患者は 2 時間に 1 回体位を変更した際、タブレットの向きを変えることで対応が可能である。また、患者が横向きに体を向けている際でもカーソルを操作できるように、小さな首の振り幅で操作が可能になっている。

表 8.1: 作業時間

	1 回目(秒)	2 回目(秒)	3 回目(秒)	平均(秒)
被験者 1	92.1	56.7	42.1	63.6
被験者 2	60.2	83.0	101.6	81.6
被験者 3	43.9	53.5	39.6	45.7
被験者 4	93.2	88.5	55.5	79.1
被験者 5	54.6	38.6	32.6	41.9

表 8.2: ヒューマンインタフェースの条件（3.2.1 節）

	被験者 1	被験者 2	被験者 3	被験者 4	被験者 5	平均
操作の検知	5	4	5	3	5	4.4
安全性	5	4	5	3	5	4.4
清潔	5	5	5	3	5	4.6
直観的	5	4	5	4	4	4.4
プライバシーの保護	2	5	2	3	5	3.4

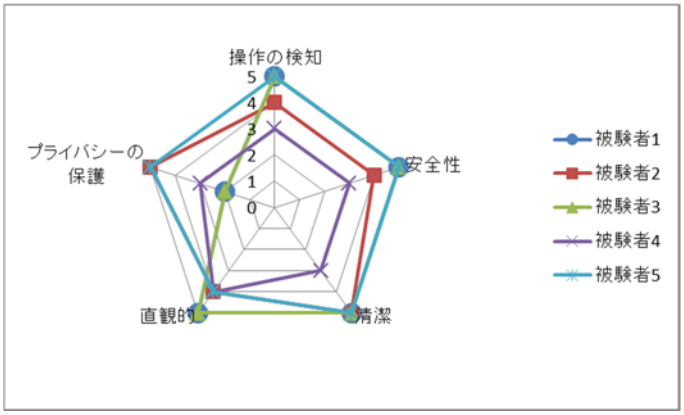


図 8.2: ヒューマンインタフェースの条件（3.2.1 節）

表 8.3: 操作盤の条件（7.4.1 節）

	被験者 1	被験者 2	被験者 3	被験者 4	被験者 5	平均
内容の把握	5	4	4	3	4	4
操作の邪魔にならない	3	3	1	3	4	2.8
人の動きと　かみあう	4	4	4	4	5	4.2

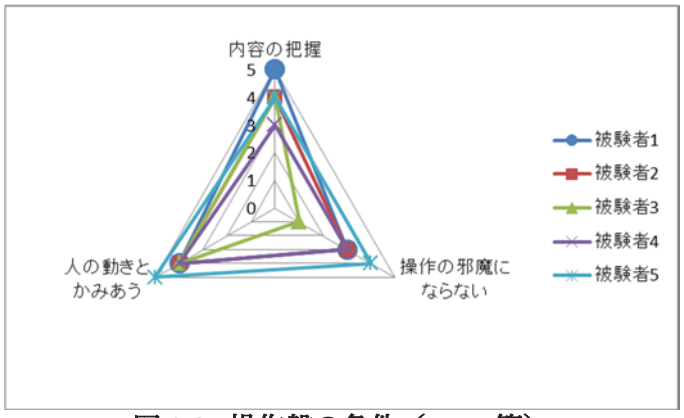


図 8.3: 操作盤の条件（7.4.1 節）

表 8.4: ユニバーサルデザイン 7 原則

	被験者 1	被験者 2	被験者 3	被験者 4	被験者 5	平均
公平性	3	4	2	3	4	3.2
柔軟性	5	4	5	4	5	4.6
身体への負荷が少ない	4	4	5	4	5	4.4

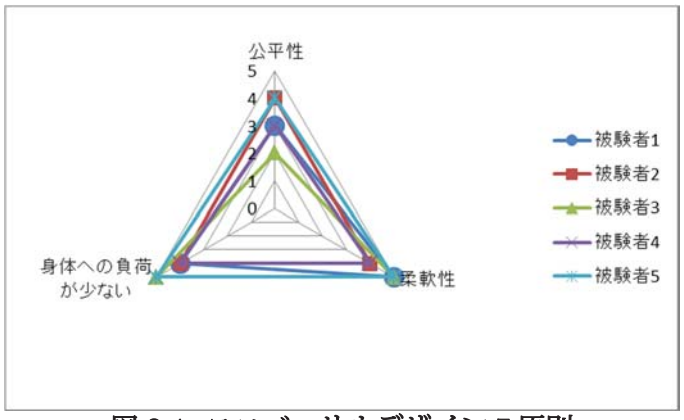


図 8.4: ユニバーサルデザイン 7 原則

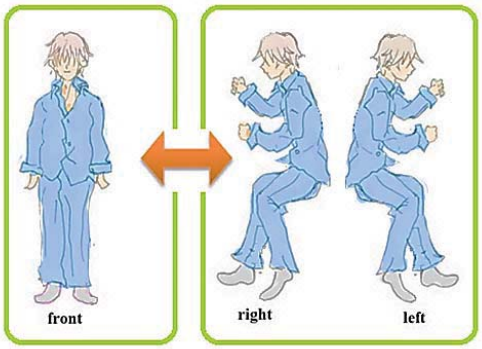


図 8.5：患者の 3 姿勢[50]

8.3 左右方向の移動速度修正

前節での実験後の被験者の意見から、左右の移動方向を検出しているオプティカルフローを削除することにした。また図 8.6 のように、移動方向を検知していた「右」領域を、現在のインターフェースの 1/2 の移動速度の「右連続」に修正を行う。「左」領域も同様の修正を行う。修正後のプログラムを付録に記す。

```
//⑤位置　右→→　「右連続」領域
else if(x0<mouth_x && mouth_x<r1x && y1<mouth_y && mouth_y<y3){

        po.x += 30;
        SetCursorPos(po.x ,po.y);


      //右矢印｜→（右向き三角と線）をピンク色で描画
      //ピンク色CV_RGB (255, 153, 160)
      //三角
        cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
        cvPoint(32 + genten_idou_x ,80 + genten_idou_y),
        CV_RGB(255, 153, 160) , 12, CV_AA, 0); //
        cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
        cvPoint(32 + genten_idou_x ,90+ genten_idou_y),
        CV_RGB (255, 153, 160) , 12, CV_AA, 0); // |
        cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
        cvPoint(32+ genten_idou_x ,90 + genten_idou_y),
        CV_RGB (255, 153, 160) , 12, CV_AA, 0); //／

        count_click = 0;
}


//⑥位置　右→　「右」領域
else if (r1x<mouth_x && mouth_x<x1 && y1<mouth_y && mouth_y<y3){

    //向き C　右→ ピンク
      po.x += 15;
      SetCursorPos(po.x ,po.y);


    //右矢印→（右向き三角）をピンク色で描画
    //ピンク色  CV_RGB (255, 153, 160)
      cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
      cvPoint(32 + genten_idou_x ,80 + genten_idou_y),
      CV_RGB(255, 153, 160) , 12, CV_AA, 0); //
      cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
      cvPoint(32 + genten_idou_x ,90+ genten_idou_y),
      CV_RGB (255, 153, 160) , 12, CV_AA, 0); // |
      cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
```

```
        cvPoint(32+ genten_idou_x ,90 + genten_idou_y),
        CV_RGB (255, 153, 160) , 12, CV_AA, 0); //／

        count_click = 0;
}
```

図 8.6: 右方向の移動速度修正

第 9 章 結言

本研究では、脊髄損傷のような身体の自由が極端に制限される患者の QOL を向上するために、息と顔の動きを用いてタブレットを操作するヒューマンインタフェースの基礎を確立し、タブレットのタッチレス化を行った。また、研究成果を学外にて発表した[51,52]。

一方で、最初に操作盤に顔を合わせる際に初期位置の設定が難しいため、顔を合わせる枠を表示し、操作盤の映像のズームと縮小機能を追加したり、また、文字を入力する際に QWERTY 配列のソフトウェアキーボードだとキーが小さくて打ちにくいため、任意で 10 キーボードを表示させる機能を追加したりする必要があるなど、本研究のインターフェースを実用化するために問題が多く残されており、今後の課題である。

謝辞

本論文の作成及び研究活動に対して、厳粛かつ温かなご指導を頂きましたグローバルエンジニアリング学部機械創造工学科金丸隆志准教授に深くお礼申し上げます。また、本論文の執筆にあたり、丁寧かつ熱心なご指導を頂きました、工学部機械システム工学科見崎大悟准教授、工学部機械システム工学科羽田靖史准教授、九州工業大学大学院情報工学研究院システム創成情報工学研究系中茎隆准教授、産学連携 ECP センター花野井利之様に心よりお礼申し上げます。

また、いつも私を支えて下さった知能機械研究室の同輩、後輩の皆様にこの場で感謝の意を表します。

最後にいつも大学院で学ぶ機会を与え、応援して下さった母に感謝致します。

参考文献

[1] 住田幹男，杉原勝宣　他「日本における高齢者脊髄損傷の状況」
日本職業・災害医学会会誌 JJOMT Vol.52 No.1, pp.17-23,2003

[2] 新宮彦助「日本における脊損発生の疫学調査　第 3 報」
日本パラプレジア医学会雑誌 8（1）, pp.26-27,1995

[3] 柴崎啓一「全国脊髄損傷登録統計」
日本脊髄障害医学会雑誌 vol.18 No.1, pp.271-274,2005

[4] 徳器技研工業株式会社
http://homepage3.nifty.com/tokuso/

[5] 株式会社日立ケーイーシステムズ
http://www.hke.jp/index.htm

[6] Tobii Technology
http://www.tobii.com/

[7] 独立行政法人労働者健康福祉機構吉備高原医療リハビリテーションセンター
http://www.kibirihah.rofuku.go.jp/index.html

[8] メルクマニュアル家庭版，脊髄 76 章神経系のしくみと働き
http://merckmanual.jp/mmhe2j/sec06/ch076/ch076c.html

[9] 大阪府　脊損についての医療的知識　脊椎と脊髄について
http://www.pref.osaka.jp/keikakusuishin/kankou/sekituitosekizui.html

[10] アイリス・アイリスの作業現場
http://web.kyoto-inet.or.jp/people/irisiris/index.html

[11] 新小文字病院脊髄脊椎外科治療センター　脊髄と脊椎について
http://center.shinkomonji-hp.jp/about/

[12] 柴崎啓一，田村睦弘「脊髄ヘルスケア・基礎編」
NPO 法人日本せきずい基金, pp.13-20,2005

[13] あばば　脊髄損傷
http://www.h5.dion.ne.jp/ababa-55/ababaotakebi5.spinalcord scirehab.sub3.html

[14] 井口哲弘，津村暢宏，加藤順一[他]　「脊損パンフレット」
兵庫県立総合リハビリテーションセンター　リハビリテーション中央病院
脊髄損傷患者さんに関する調査委員会，pp.1-22，2006

[15] 加藤文彦「非骨傷性頸髄損傷予防法と早期治療体系の確立」
独立行政法人　労働者健康福祉機構
勤労者脊椎，脊髄損傷研究センター，pp.1-19，2009

[16] 猪川輪哉，植田尊善「上肢機能、歩行機能の神経学的診断」
「脊椎脊髄ジャーナル」2003,4 月号

[17] 日本唯一の脊髄損傷専門トレーニング施設「J-Workout」、
車いすスポーツに特化した脊髄損傷者リハビリプログラム
http://www.dreamnews.jp/press/0000012280/

[18] ジェイ・ワークアウト株式会社
http://www.j-workout.com/

[19] 脊椎損傷、脊髄損傷　家庭の医学- Yahoo!ヘルスケア
http://health.yahoo.co.jp/katei/detail/ST170060/2/

[20] 松本琢磨「身体障害者用補助具（マウススティックや固定台等）」
OT ジャーナル 46, pp.846-850,2012

[21] The Eye Tribe
https://theeyetribe.com/

[22] 廣田祐樹「マウススティック作成マニュアル」
SCIOT（脊髄損傷作業療法研究会）神奈川リハビリテーション病院

[23] 歯科大辞典【歯医者/歯科情報の歯チャンネル】
http://www.ha-channel-88.com/jiten/

[24] 日本経済新聞
http://www.nikkei.com/
http://www.nikkei.com/article/DGXNASFK1600V_W3A110C1000000/

[25] ZMP
http://www.zmp.co.jp/?lang=jp

[26] Microsoft
http://www.microsoft.com/ja-jp/default.aspx

[27] 中根正樹，森永俊彦　他
「気管吸引ガイドライン 2013（成人で人工気道を有する患者のための）」
日本呼吸療法医学会気管吸引ガイドライン改訂ワーキンググループ
人工呼吸　Jpn J Respir Care 30, pp.75-91,2013

[28] 吉川英史，中茎隆「息圧センサを用いた脊髄損傷患者向けヒューマンインタフェースの検討」
電気学会研究会資料　IIS，次世代産業システム研究会
（The papers of Technical Meeting on Innovative Industrial System），
IEEE Japan 2013(2), pp. 1-5, 2013

[29] 吉川英史「脊髄損傷患者に適したヒューマンインタフェースの開発」
工学院大学　学士論文,2012

[30] Freescale Semiconductor
http://www.freescale.com/files/sensors/doc/data_sheet/MPXV5004G.pdf

[31] Pico Technology
http://www.picotech.com/

[32] 調査リポート：2013 年の世界タブレット市場、Apple はシェアを落とすも依然として圧倒的首位――Gartner 調べ - ITmedia Mobile
http://www.itmedia.co.jp/mobile/articles/1403/04/news060.html

[33] スイッチサイエンス
https://www.switch-science.com

[34] Smart Projects
http://www.smartprj.com/catalog/index.php

[35] 永田雅人, 豊沢聡「実践 OpenCV―映像処理&解析」
カットシステム,2-3,10-11,149-165,182-197,2009

[36] IT 用語辞典 e-Words
http://e-words.jp/

[37] 浮動小数点数型と誤差
http://www.cc.kyoto-su.ac.jp/~yamada/programming/float.html#floating

[38] Bruce. D. Lucas and Takeo. Kanade, “An Iterative Image Registration Technique with an Application to Stereo Vision,” Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI 81), pp.674~679, 1981

[39] Jean-yves Bouguet, “Pyramidal implementation of the lucas kanade feature tracker, Description of the algorithm”, Technical report, Intel Corporation, Microprocessor Research Labs, 1999

[40] 人間特性情報リンク集
独立行政法人製品評価技術基盤機構
http://www.nite.go.jp/jiko/s_standard/human_db_link/index.html

[41] NITE 人間特性データベース
独立行政法人　製品評価技術基盤機構（NITE）
http://www.tech.nite.go.jp/human/

[42] Paul Viola and Michael J. Jones,“Rapid Object Detection using a Boosted Cascade of Simple Features”, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2001 (CVPR 2001),Vol.1,pp.511~518, 2001

[43] Rainer Lienhart and Jochen Maydt, “An Extended Set of Haar-like Features for Rapid Object Detection”, Proceedings of the IEEE International Conference on Image Processing 2002(ICIP 2002),Vol.1,pp.900-903,2002

[44]伊藤 啓 他「カラーユニバーサルデザイン推奨配色セット」,
NPO 法人カラーユニバーサルデザイン機構, pp.1~12, 2013

[45] のぶしま矯正歯科
http://www.nob-ortho.or.jp

[46] Gary Bradski, Adrian Kaehler, 松田 晃一(翻訳)
「詳解 OpenCV ―コンピュータビジョンライブラリを使った画像処理・認識」
オライリージャパン,pp.322-341, 2009

[47] 9 月 OS シェア、Windows 7 が増加 | マイナビニュース(Net Applications)
http://news.mynavi.jp/news/2014/10/04/028/

[48] ASCII.jp ビジネス
http://ascii.jp/elem/000/000/884/884449/

[49] 北原 義典「イラストで学ぶ ヒューマンインタフェース」
講談社,pp.136-137,pp.184-185,2011

[50] 谷島綾乃, 中茎隆「脊髄損傷患者のための水飲み器の製作」
第 13 回計測自動制御学会システムインテグレーション部門講演会(SI2012)
2M3-2, 福岡, pp.1871-1873,2012

[51]Ayano Tanishima , Takashi Nakakuki, Hiroto Noguchi and Takashi Kanamaru, "Development of drinking support device for spinal cord injuries to enhance quality of life",2013 13th International Conference on Control, Automation and Systems (ICCAS2013), pp.1459-1462 , 2013

[52]Ayano Tanishima and Takashi Kanamaru, "Development of human interface for spinal cord injury patient to enhance quality of life" The 1st Innovation Forum of Advanced Engineering and Education (IFAEE), pp.107 ,2014

付録

Ⅰ　Arduino のプログラム（Arduino）

ⅰ－1 Arduino による息の判定プログラム

```
1   #include <Max3421e.h>
2   #include <Usb.h>
3   #include <AndroidAccessory.h>
4
5
6   // スイッチのポートを定義
7   #define POTENTIOMETER A0
8
9   //外部インテントのための指定
10  //AndroidAccessoryオブジェクトを作成
11  AndroidAccessory acc("MyManufacturer",              // 製造者名
12  "MyModel",                                  // モデル名
13  "This is a sample accessory",   // 説明文
14  "1.0",                                      // バージョン名
15  "http://android.com",                 // URL
16  "0000000012345678");                  // シリアル番号
17
18
19
20  // メッセージバッファ
21  byte msg[1];//変数msgを定義変
22  byte previous=0; //変数をprevious定義
23
24  //kaisuuという変数を定義。
25  int kaisuu = 0;
26  int global_count = 0;
27
28
29
30  void setup() //最初に一度だけ実行される部分
31  {
32     Serial.begin(115200); // 115200bpsでシリアル通信のポートを開きます
33     Serial.print("¥r¥nStart");
34
35     pinMode(0, INPUT); // A0番ピン圧力センサからの入力に設定
36     pinMode(13, OUTPUT); // 13番ピンを出力に設定
37
```

```
38      acc.powerOn();//USBホスト機能を有効にする
39  }
40
41
42
43  void loop()   //繰り返し実行される部分
44  {
45      int status;
46      if (acc.isConnected()) { //Androidを起動・接続する命令を送る
47        // アナログ値を読み込む
48        status = analogRead(0);
49
50        //リセット
51        if(global_count>15){
52        //   1500ms = 100ms * 15回, global_countが15回以下ならばglobal_countと
53        //kaisuuを0にする
54          global_count = 0;
55          kaisuu = 0;
56        }
57
58        msg[0] = 0;//msg変数を0と置く
59        if (status >= 247.7){
60        //しきい値 : 測定電圧1.21065（Ｖ）÷基準電圧(電源電圧)（５Ｖ）
61        //X  1023(変換段階数)
62          ＝ A/D変換出力値（しきい値）247.7
63        //しきい値が2回以上超える
64          if (kaisuu < 2) {
65            kaisuu=kaisuu+1;
66
67          } else{ //  しきい値より大きくなった&2回以上なら以下を処理
68            msg[0] = 1;//しきい値を超えたらmsg変数を1にする
69
70            kaisuu = 0;
71            global_count = 0;
72          }
73        }
74
75
76        // Androidに送信
77        if(msg[0] != previous){//msgとpreviousが等しくない(!=)ならば:時刻n
78          acc.write(msg, sizeof(msg));   //  ※ここに注意
79        }
80
81        if(msg[0] == 1){//msgが1ならばLEDが点灯
82          digitalWrite(13, HIGH) ; // LEDを点灯(HIGH:5V)で出力
83          delay(100);                          // 300msec待つ
84          digitalWrite(13, LOW);       // LEDpin8に0Vを出力
85        }
86
87        //LED点灯が終わったら下記の命令を行こなう
88        // Androidに送信
89        if(msg[0] != previous){//msgとpreviousが等しくない(!=)ならば（時刻nのとき
90
```

```
91              acc.write(msg, sizeof(msg));   // ※ここに注意
92          }
93
94          //LED点灯が終わったら下記の命令を行こなう
95          previous = msg[0];//previous = msg  時刻n+1
96          global_count = global_count + 1;//global_countを1増やす
97      }
98      delay(100);
99  }
100
101
```

ｉ－2 Arduino による Windows タブレットのスリープ解除プログラム（マウスクリック）

```
#include <Max3421e.h>
#include <Usb.h>


// スイッチのポートを定義
#define POTENTIOMETER A0

// メッセージバッファ
byte msg[1];//変数msgを定義変
byte previous=0; //変数をprevious定義

//kaisuuという変数を定義。
int kaisuu = 0;
int global_count = 0;

//カーソル移動の為の変数を定義
int full = 100;
int radius = 100;
int delay_msec = 2;

int phase = 0;
int x = radius;
int y = 0;



void setup() //最初に一度だけ実行される部分
{
  pinMode(0, INPUT); // A0番ピン圧力センサからの入力に設定
  pinMode(13, OUTPUT); // 13番ピンを出力に設定

  Mouse.begin();//マウスライブラリを開始する
}


void loop()   //繰り返し実行される部分
{
  int status;
  // アナログ値を読み込む
  status = analogRead(0);

  //リセット
  if(global_count>15){
  //   1500ms = 100ms * 15回, global_countが15回以下ならば
 //global_countとkaisuuを0にする
     global_count = 0;
```

```
48        kaisuu = 0;
49      }
50
51
52      msg[0] = 0;//msg変数を0と置く
53      if (status >= 247.7){
54      //しきい値 ： 測定電圧1.21065（Ｖ）÷基準電圧(電源電圧)（５Ｖ）
55      //Ｘ 1023(変換段階数) ＝ A/D変換出力値（しきい値）247.7
56      //しきい値が2回以上超える
57        if (kaisuu < 2) {
58          kaisuu = kaisuu+1;
59        }else{ //  しきい値より大きくなった&2回以上なら以下を処理
60          msg[0] = 1;//しきい値を超えたらmsg変数を1にする
61
62          kaisuu = 0;
63          global_count = 0;
64        }
65      }
66
67      if(msg[0] == 1){//msgが1ならばLEDが点灯
68        digitalWrite(13, HIGH) ; // LEDを点灯(HIGH:5V)で出力
69        delay(100);                              // 300msec待つ
70        digitalWrite(13, LOW);       // LEDpin8に0Vを出力
71      }
72
73      //LED点灯が終わったら下記の命令を行こなう
74      //  タブレットに送信
75      if(msg[0] != previous && msg[0]==1){//msgとpreviousが等しくない(!=)、
76        /*
77      かつprevious0⇒1（msg[0] != previousではprevious0⇒1とprevious1⇒0、
78      すなわち、しきい値が超えたときと戻る時を検出してしまうのでmsg[0]==1の時
79    のみ以下を行う）ならば（時刻nのとき
80      */
81
82        Mouse.click();//右クリックを送信する。
83    // Mouse.move(10, 10, 0); //マウスを動かす
84      }
85
86      //LED点灯が終わったら下記の命令を行こなう
87      previous = msg[0];//previous = msg  時刻n+1
88      global_count = global_count + 1;//global_countを1増やす
89
90      delay(100); //そのまま  0.1  秒待つ。
91    }
92
93
```

i －3 タクトスイッチによるマウカーソルの操作プログラム

```
1   // 定数の定義
2   const int LED_up = 8;
3   const int LED_down = 9;
4   const int LED_left = 10;
5   const int LED_right = 11;
6   const int LED_click = 12;
7
8   const int BUTTON_up = 2;
9   const int BUTTON_down = 3;
10  const int BUTTON_left = 4;
11  const int BUTTON_right = 5;
12  const int BUTTON_click = 6;
13
14  //スイッチの回路は負論理なので、ON が LOW になる
15  const int SW_ON = LOW;
16  const int SW_OFF = HIGH;
17
18  // フラグの定義
19  const int FLG_ON = 1;
20  const int FLG_OFF = 0;
21
22  int G_oldval_up = SW_OFF;                  // スイッチの状態 （初期設定は OFF)
23  int G_oldval_down = SW_OFF;
24  int G_oldval_left = SW_OFF;
25  int G_oldval_right = SW_OFF;
26  int G_oldval_click = SW_OFF;
27
28  int G_state_up =   FLG_OFF;                // LED 点滅の状態 （初期設定は OFF）
29  int G_state_down =   FLG_OFF;
30  int G_state_left =   FLG_OFF;
31  int G_state_right =   FLG_OFF;
32  int G_state_click =   FLG_OFF;
33
34
35
36  //最初に 1 回だけ実行する処理
37  void setup() {
38
39    pinMode( BUTTON_up, INPUT );      // 2 番ピン圧力センサからの入力に設定
40    pinMode( BUTTON_down, INPUT );      // 3 番ピン圧力センサからの入力に設定
41    pinMode( BUTTON_left, INPUT );      // 4 番ピン圧力センサからの入力に設定
42    pinMode( BUTTON_right, INPUT );      // 5 番ピン圧力センサからの入力に設定
43    pinMode( BUTTON_click, INPUT );      // 6 番ピン圧力センサからの入力に設定
44    //停止　白ボタン　LED 赤
45
46    pinMode( LED_up, OUTPUT );         // 8 番ピンを出力に設定
47    pinMode( LED_down, OUTPUT );         // 9 番ピンを出力に設定
48    pinMode( LED_left, OUTPUT );         // 10 番ピンを出力に設定
49
```

```
50      pinMode( LED_right, OUTPUT );          // 11 番ピンを出力に設定
51      pinMode( LED_click, OUTPUT );          // 12 番ピンを出力に設定
52  }
53
54
55  //繰り返し実行する処理
56  void loop() {
57      // スイッチの状態を読み取る
58      int val_up = digitalRead(BUTTON_up);
59      int val_down = digitalRead(BUTTON_down);
60      int val_left = digitalRead(BUTTON_left);
61      int val_right = digitalRead(BUTTON_right);
62      int val_click = digitalRead(BUTTON_click);
63
64
65  //上
66      //スイッチが ON、前回が OFF だったとき
67      if(val_up == SW_ON && G_oldval_up == SW_OFF) {
68        //LED 点滅の状態を切り替える
69        if(G_state_up == FLG_ON){      // フラグが ON のとき
70              G_state_up = FLG_OFF;            // OFF
71          } else {                          // フラグが OFF のとき
72              G_state_up = FLG_ON;             // ON
73          }
74      }
75         G_oldval_up = val_up;                  // スイッチの状態を G_oldval_up へ保存
76
77  //LED の点滅状態を開始するとき
78  if (G_state_up == FLG_ON){           // LED の点滅状態が ON のとき
79        digitalWrite(LED_up, SW_ON);       // 緑色 LED 点灯
80        Mouse.move(0, -1, 0);
81        //マウス上にを動かす　(x, y, wheel マウスホイールをスクロールする量)
82
83        //他のスイッチを切る
84          digitalWrite(LED_down, SW_OFF);
85          digitalWrite(LED_left, SW_OFF);
86          digitalWrite(LED_right, SW_OFF);
87          digitalWrite(LED_click, SW_OFF);
88
89       //フラグをリセット
90        G_state_down = FLG_OFF;
91        G_state_left = FLG_OFF;
92        G_state_right = FLG_OFF;
93        G_state_click = FLG_OFF;
94
95           } else　{ // OFF のとき
96          digitalWrite( LED_up, SW_OFF ); // LED を消灯
97          }
98
99
100
101  //下
102   //スイッチが ON、前回が OFF だったとき
```

```
  if(val_down == SW_ON && G_oldval_down == SW_OFF) {
    //LED 点滅の状態を切り替える
    if(G_state_down == FLG_ON){      // フラグが ON のとき
          G_state_down = FLG_OFF;            // OFF
      } else {                         // フラグが OFF のとき
          G_state_down = FLG_ON;             // ON
      }
  }
     G_oldval_down = val_down;            // スイッチの状態を G_oldval_down へ保存

//LED の点滅状態を開始するとき
if (G_state_down == FLG_ON){           // LED の点滅状態が ON のとき
    digitalWrite(LED_down, SW_ON);      // 黄色 LED 点灯
    Mouse.move(0, 1, 0); //マウス下にを動かす

    //他のスイッチを切る
      digitalWrite(LED_up, SW_OFF);
      digitalWrite(LED_left, SW_OFF);
      digitalWrite(LED_right, SW_OFF);
      digitalWrite(LED_click, SW_OFF);

   //フラグをリセット
    G_state_up = FLG_OFF;
    G_state_left = FLG_OFF;
    G_state_right = FLG_OFF;
    G_state_click = FLG_OFF;

      } else   { // OFF のとき
      digitalWrite( LED_down, SW_OFF ); // LED を消灯
      }


  //左
  //スイッチが ON、前回が OFF だったとき
  if(val_left == SW_ON && G_oldval_left == SW_OFF) {
    //LED 点滅の状態を切り替える
    if(G_state_left == FLG_ON){      // フラグが ON のとき
          G_state_left = FLG_OFF;            // OFF
      } else {                         // フラグが OFF のとき
          G_state_left = FLG_ON;             // ON
      }
  }
     G_oldval_left = val_left;                // スイッチの状態を G_oldval_left へ保存

//LED の点滅状態を開始するとき
if (G_state_left == FLG_ON){           // LED の点滅状態が ON のとき
    digitalWrite(LED_left, SW_ON);      // 青色 LED 点灯
    Mouse.move(-1, 0, 0); //マウス左←にを動かす

    //他のスイッチを切る
      digitalWrite(LED_up, SW_OFF);
      digitalWrite(LED_down, SW_OFF);
```

```
156         digitalWrite(LED_right, SW_OFF);
157         digitalWrite(LED_click, SW_OFF);
158
159     //フラグをリセット
160       G_state_up = FLG_OFF;
161       G_state_down = FLG_OFF;
162       G_state_right = FLG_OFF;
163       G_state_click = FLG_OFF;
164
165          } else   { // OFF のとき
166         digitalWrite( LED_left, SW_OFF ); // LED を消灯
167         }
168
169
170
171    //右
172     //スイッチが ON、前回が OFF だったとき
173    if(val_right == SW_ON && G_oldval_right == SW_OFF) {
174       //LED 点滅の状態を切り替える
175       if(G_state_right == FLG_ON){      // フラグが ON のとき
176             G_state_right = FLG_OFF;            // OFF
177          } else {                          // フラグが OFF のとき
178             G_state_right = FLG_ON;             // ON
179          }
180    }
181        G_oldval_right = val_right;            // スイッチの状態を G_oldval_right へ保存
182
183 //LED の点滅状態を開始するとき
184 if (G_state_right == FLG_ON){           // LED の点滅状態が ON のとき
185       digitalWrite(LED_right, SW_ON);      // ピンク色 LED 点灯
186       Mouse.move(1, 0, 0); //マウス右→にを動かす
187
188       //他のスイッチを切る
189          digitalWrite(LED_up, SW_OFF);
190          digitalWrite(LED_down, SW_OFF);
191          digitalWrite(LED_left, SW_OFF);
192          digitalWrite(LED_click, SW_OFF);
193
194     //フラグをリセット
195       G_state_up = FLG_OFF;
196       G_state_down = FLG_OFF;
197       G_state_left = FLG_OFF;
198       G_state_click = FLG_OFF;
199
200          } else   { // OFF のとき
201         digitalWrite( LED_right, SW_OFF ); // LED を消灯
202         }
203
204
205     //クリック
206     //クリックは 1 回のみ
207    //スイッチが ON、前回が OFF だったとき
208
```

```
209     if(val_click == SW_ON && G_oldval_click == SW_OFF) {
210       //LED 点滅の状態を切り替える
211       if(G_state_click == FLG_ON){       // フラグが ON のとき
212             G_state_click = FLG_OFF;              // OFF
213           } else {                              // フラグが OFF のとき
214             G_state_click = FLG_ON;               // ON
215           }
216     }
217         G_oldval_click = val_click;               // スイッチの状態を G_oldval_down へ保存
218
219   //LED の点滅状態を開始するとき
220   if (G_state_click == FLG_ON){             // LED の点滅状態が ON のとき
221       digitalWrite(LED_click, SW_ON);       // 赤色 LED 点灯
222       Mouse.click();//右クリックを送信する
223
224       delay(100);
225       digitalWrite(LED_click, SW_OFF);
226
227     //他のスイッチを切る
228       digitalWrite(LED_up, SW_OFF);
229       digitalWrite(LED_down, SW_OFF);
230       digitalWrite(LED_left, SW_OFF);
231       digitalWrite(LED_right, SW_OFF);
232
233     //フラグをリセット
234       G_state_up = FLG_OFF;
235       G_state_down = FLG_OFF;
236       G_state_left = FLG_OFF;
237       G_state_right = FLG_OFF;
238       G_state_click = FLG_OFF;
239
240     } else   { // OFF のとき
241     digitalWrite( LED_click, SW_OFF ); // LED を消灯
242     }
243     delay(10);          // チャタリング防止のウエイト
244   }
245
246
247
248
```

ｉ－4 息圧センサによるマウカーソルの移動プログラム

```
1  #include <Max3421e.h>
2  #include <Usb.h>
3
4
5  const int LED_up = 8;
6  const int LED_down = 9;
7  const int LED_left = 10;
8  const int LED_right = 11;
9  const int LED_click = 12;
10
11 const int BUTTON_up = 0;
12 const int BUTTON_down = 1;
13 const int BUTTON_left = 2;
14 const int BUTTON_right = 3;
15 const int BUTTON_click = 4;
16
17 const int SW_ON = 1;
18 const int SW_OFF = 0;
19
20 // フラグの定義
21 const int FLG_ON = 1;
22 const int FLG_OFF = 0;
23
24
25 //息圧判定
26 // メッセージバッファ
27 byte msg_up[1];//変数msgを定義変
28 byte msg_down[1];
29 byte msg_left[1];
30 byte msg_right[1];
31 byte msg_click[1];
32
33 byte msg_old_up = SW_OFF;          // スイッチの状態 （初期設定はOFF)。前回の情
34 報。
35 byte msg_old_down = SW_OFF;       //変数をmsg_oldに定義
36 byte msg_old_left = SW_OFF;
37 byte msg_old_right = SW_OFF;
38 byte msg_old_click = SW_OFF;
39
40 int G_state_up =    FLG_OFF;      // LED点滅の状態 （初期設定はOFF）
41 int G_state_down =    FLG_OFF;
42 int G_state_left =    FLG_OFF;
43 int G_state_right =    FLG_OFF;
44 int G_state_click =    FLG_OFF;
45
46
47
48 //kaisuuという変数を定義。
49 int kaisuu_up = 0;
```

```
int kaisuu_down = 0;
int kaisuu_left = 0;
int kaisuu_right = 0;
int kaisuu_click = 0;

int global_count_up = 0;
int global_count_down = 0;
int global_count_left = 0;
int global_count_right = 0;
int global_count_click = 0;

/*
しきい値を定義
しきい値 ： 測定電圧1.21065（Ｖ）÷基準電圧(電源電圧)（５Ｖ）
Ｘ 1023(変換段階数) ＝ A/D変換出力値（しきい値）247.7
*/
int Shikiiti = 247.7;



//最初に1回だけ実行する処理
void setup() {
 //入力
   pinMode( BUTTON_up, INPUT );      // A0番ピン圧力センサからの入力に設定
   pinMode( BUTTON_down, INPUT );   // A1番ピン圧力センサからの入力に設定
   pinMode( BUTTON_left, INPUT );   // A2番ピン圧力センサからの入力に設定
   pinMode( BUTTON_right, INPUT ); // A3番ピン圧力センサからの入力に設定
   pinMode( BUTTON_click, INPUT ); // A4番ピン圧力センサからの入力に設定
 //停止　白ボタン　LED赤

 // 出力
   pinMode( LED_up, OUTPUT );          // 8番ピンを出力に設定
   pinMode( LED_down, OUTPUT );      // 9番ピンを出力に設定
   pinMode( LED_left, OUTPUT );      // 10番ピンを出力に設定
   pinMode( LED_right, OUTPUT );     // 11番ピンを出力に設定
   pinMode( LED_click, OUTPUT );     // 12番ピンを出力に設定

   Mouse.begin();   //マウスライブラリを開始する
   Keyboard.begin(); // キーボードの制御を初期化する。
}


//繰り返し実行する処理
void loop() {
   // スイッチの状態を読み取る
   int val_up       = analogRead(BUTTON_up);
   int val_down    = analogRead(BUTTON_down);
   int val_left    = analogRead(BUTTON_left);
   int val_right = analogRead(BUTTON_right);
   int val_click = analogRead(BUTTON_click);
```

```
//上↑
//息圧判定
    //リセット
     if(global_count_up > 15){
//1500ms = 100ms * 15回, global_countが15回以下ならば
//global_countとkaisuuを0にする
          global_count_up = 0;
          kaisuu_up = 0;
          }

  msg_up[0] = 0;//msg変数を0と置く.変数の初期化
  if (val_up >= Shikiiti){
/*
しきい値 : 測定電圧1.21065（V）÷基準電圧(電源電圧)（5 V）
X 1023(変換段階数) = A/D変換出力値（しきい値）247.7
*/
//しきい値が2回以上超える
     if (kaisuu_up < 2) {
        kaisuu_up = kaisuu_up + 1;
     } else { //  しきい値より大きくなった&2回以上なら以下を処理
        msg_up[0] = 1;//しきい値を超えたらmsg変数を1にする

        kaisuu_up = 0;
        global_count_up = 0;
     }
  }


//息圧判定により息と判定されスイッチがONのとき、かつ前回は息がはいってない
//（スイッチがOFF）のとき
//  タブレットに送信
  if(msg_up[0] != msg_old_up && msg_up[0]==1){
/*
msgとmsg_up(previous)が等しくない(!=)、
かつprevious0⇒1（msg[0] != previousではprevious0⇒1とprevious1⇒0、
すなわち、しきい値が超えたときと戻る時を検出してしまうのでmsg[0]==1の時のみ
以下を行う）ならば（時刻nのとき
*/

//スイッチがON、前回がOFFだったとき
//LED点滅の状態を切り替える
     if(G_state_up == FLG_ON){      //  フラグがONのとき
          G_state_up = FLG_OFF;            // OFF
        } else {                          //  フラグがOFFのとき
          G_state_up = FLG_ON;             // ON
        }

  }
    msg_old_up = msg_up[0];                  //  スイッチの状態をG_oldval_upへ保存

//LEDの点滅状態を開始するとき
```

```
156 if (G_state_up == FLG_ON){            // LEDの点滅状態がONのとき
157       digitalWrite(LED_up, SW_ON);       // 緑色LED点灯
158       Mouse.move(0, -1, 0);
159  //マウス上にを動かす　(x, y, wheel マウスホイールをスクロールする量)
160
161       //他のスイッチを切る
162          digitalWrite(LED_down, SW_OFF);
163          digitalWrite(LED_left, SW_OFF);
164          digitalWrite(LED_right, SW_OFF);
165          digitalWrite(LED_click, SW_OFF);
166
167      //フラグをリセット
168       G_state_down = FLG_OFF;
169       G_state_left = FLG_OFF;
170       G_state_right = FLG_OFF;
171       G_state_click = FLG_OFF;
172
173
174       //msg_old_up = msg_up[0];//previous_up = msg_up 時刻n+1
175       global_count_up = global_count_up + 1;//global_count_upを1増やす
176
177           } else   { // OFFのとき
178          digitalWrite( LED_up, SW_OFF ); // LEDを消灯
179          }
180
181 //下↓
182 //息圧判定
183      //リセット
184       if(global_count_down > 15){
185 //    1500ms = 100ms * 15回, global_countが15回以下ならば
186 //global_countとkaisuuを0にする
187            global_count_down = 0;
188            kaisuu_down = 0;
189            }
190
191    msg_down[0] = 0;//msg変数を0と置く.変数の初期化
192    if (val_down >= Shikiiti){
193 /*
194 しきい値 : 測定電圧1.21065（V）÷基準電圧(電源電圧)（5 V）
195 X  1023(変換段階数) ＝ A/D変換出力値（しきい値）247.7
196 */
197       //しきい値が2回以上超える
198       if (kaisuu_down < 2) {
199          kaisuu_down = kaisuu_down + 1;
200       } else {//  しきい値より大きくなった&2回以上なら以下を処理
201          msg_down[0] = 1;//しきい値を超えたらmsg変数を1にする
202
203          kaisuu_down = 0;
204          global_count_down = 0;
205       }
206    }
207
208
```

```
209
210 //息圧判定により息と判定されスイッチがONのとき、
211 //かつ前回は息がはいってない（スイッチがOFF）のとき
212 //  タブレットに送信
213    if(msg_down[0] != msg_old_down && msg_down[0]==1){
214 /*
215 msgとmsg_down(previous)が等しくない(!=)、
216 かつprevious0⇒1（msg[0] != previousではprevious0⇒1とprevious1⇒0、
217 すなわち、しきい値が超えたときと戻る時を検出してしまうのでmsg[0]==1の時のみ
218 以下を行う）ならば（時刻nのとき
219 */
220    //スイッチがON、前回がOFFだったとき
221
222
223          //LED点滅の状態を切り替える
224       if(G_state_down == FLG_ON){       //  フラグがONのとき
225             G_state_down = FLG_OFF;              // OFF
226          } else {                              //  フラグがOFFのとき
227             G_state_down = FLG_ON;               // ON
228          }
229    }
230      msg_old_down = msg_down[0];                 //  スイッチの状態をG_oldval_upへ保存
231
232
233 //LEDの点滅状態を開始するとき
234 if (G_state_down == FLG_ON){            // LEDの点滅状態がONのとき
235       digitalWrite(LED_down, SW_ON);       //  緑色LED点灯
236       Mouse.move(0, 1, 0); //マウス下にを動かす
237
238       //他のスイッチを切る
239         digitalWrite(LED_up, SW_OFF);
240         digitalWrite(LED_left, SW_OFF);
241         digitalWrite(LED_right, SW_OFF);
242         digitalWrite(LED_click, SW_OFF);
243
244       //フラグをリセット
245         G_state_up = FLG_OFF;
246         G_state_left = FLG_OFF;
247         G_state_right = FLG_OFF;
248         G_state_click = FLG_OFF;;
249
250         global_count_down = global_count_down + 1;//global_count_upを1増やす
251
252  } else {   // OFFのとき
253         digitalWrite( LED_down, SW_OFF ); // LEDを消灯
254  }
255
256
257
258 //左←
259 //息圧判定
260 //リセット
261 if(global_count_left > 15){
```

```
//1500ms = 100ms * 15回, global_countが15回以下ならばglobal_countと
//kaisuuを0にする
        global_count_left = 0;
        kaisuu_left = 0;
}

msg_left[0] = 0;//msg変数を0と置く.変数の初期化
if (val_left >= Shikiiti){
/*
しきい値 : 測定電圧1.21065（V）÷基準電圧(電源電圧)（５V）
X 1023(変換段階数) = A/D変換出力値（しきい値）247.7
しきい値が2回以上超える
*/
if (kaisuu_left < 2) {
    kaisuu_left = kaisuu_left + 1;
} else { //  しきい値より大きくなった&2回以上なら以下を処理
    msg_left[0] = 1;//しきい値を超えたらmsg変数を1にする

    kaisuu_left = 0;
    global_count_left = 0;
     }
}


//息圧判定により息と判定されスイッチがONのとき、かつ前回は息がはいってない
//（スイッチがOFF）のとき
//  タブレットに送信
if(msg_left[0] != msg_old_left && msg_left[0]==1){
/*
msgとpreviousが等しくない(!=)、かつmsg_old_left0⇒1
（msg[0] != msg_old_leftではmsg_old_left0⇒1とmsg_old_left1⇒0、
すなわち、しきい値が超えたときと戻る時を検出してしまうので
msg[0]==1の時のみ以下を行う）ならば
（時刻nのときスイッチがON、前回がOFFだったとき
*/
//LED点滅の状態を切り替える
      if(G_state_left == FLG_ON){      //  フラグがONのとき
        G_state_left = FLG_OFF;             // OFF
         } else {                          //  フラグがOFFのとき
        G_state_left = FLG_ON;              // ON
         }
  }
msg_old_left = msg_left[0];                  //  スイッチの状態をG_oldval_upへ保存

//LEDの点滅状態を開始するとき
if (G_state_left == FLG_ON){           // LEDの点滅状態がONのとき
      digitalWrite(LED_left, SW_ON);      //  青色LED点灯
      Mouse.move(-1, 0, 0); //マウス左←にを動かす

      //他のスイッチを切る
      digitalWrite(LED_up, SW_OFF);
```

```
315         digitalWrite(LED_down, SW_OFF);
316         digitalWrite(LED_right, SW_OFF);
317         digitalWrite(LED_click, SW_OFF);
318
319         //フラグをリセット
320         G_state_up = FLG_OFF;
321         G_state_down = FLG_OFF;
322         G_state_right = FLG_OFF;
323         G_state_click = FLG_OFF;
324
325         global_count_left = global_count_left + 1;//global_count_upを1増やす
326
327 } else  { // OFFのとき
328         digitalWrite( LED_left, SW_OFF ); // LEDを消灯
329 }
330
331
332 //右→
333 //息圧判定
334 //リセット
335 if(global_count_right > 15){
336 //1500ms = 100ms * 15回, global_countが15回以下ならば
337 //global_countとkaisuuを0にする
338     global_count_right = 0;
339     kaisuu_right = 0;
340 }
341
342    msg_right[0] = 0;//msg変数を0と置く.変数の初期化
343
344 if (val_right >= Shikiiti){
345 /*
346 しきい値 : 測定電圧1.21065（V）÷基準電圧(電源電圧)（5 V）
347 X  1023(変換段階数) ＝  A/D変換出力値（しきい値）247.7
348 しきい値が2回以上超える
349 */
350         if (kaisuu_right < 2) {
351           kaisuu_right = kaisuu_right + 1;
352         } else { //  しきい値より大きくなった&2回以上なら以下を処理
353           msg_right[0] = 1;//しきい値を超えたらmsg変数を1にする
354
355           kaisuu_right = 0;
356           global_count_right = 0;
357         }
358 }
359
360
361 /*
362 息圧判定により息と判定されスイッチがONのとき、
363 かつ前回は息がはいってない（スイッチがOFF）のとき
364 タブレットに送信
365 */
366 if(msg_right[0] != msg_old_right && msg_right[0]==1){
367
```

```
/*
msgとmsg_right(previous)が等しくない(!=)、
かつprevious0⇒1（msg[0] != previousではprevious0⇒1とprevious1⇒0、
すなわち、しきい値が超えたときと戻る時を検出してしまうので
msg[0]==1の時のみ以下を行う）ならば（時刻nのとき
スイッチがON、前回がOFFだったとき
*/

//LED点滅の状態を切り替える
      if(G_state_right == FLG_ON){       //  フラグがONのとき
          G_state_right = FLG_OFF;              // OFF
      } else {                              //  フラグがOFFのとき
          G_state_right = FLG_ON;               // ON
      }
 }
      msg_old_right = msg_right[0];          //  スイッチの状態をG_oldval_rightへ保存


//LEDの点滅状態を開始するとき
if (G_state_right == FLG_ON){             // LEDの点滅状態がONのとき
      digitalWrite(LED_right, SW_ON);       //  緑色LED点灯
      Mouse.move(1, 0, 0); //マウス右→にを動かす

      //他のスイッチを切る
      digitalWrite(LED_up, SW_OFF);
      digitalWrite(LED_down, SW_OFF);
      digitalWrite(LED_left, SW_OFF);
      digitalWrite(LED_click, SW_OFF);

      //フラグをリセット
      G_state_up = FLG_OFF;
      G_state_down = FLG_OFF;
      G_state_left = FLG_OFF;
      G_state_click = FLG_OFF;

      global_count_right = global_count_right + 1;//global_count_rightを1増やす

} else   { // OFFのとき
      digitalWrite( LED_right, SW_OFF ); // LEDを消灯
}

//クリック
//息圧判定
//リセット
if(global_count_click > 15){
//    1500ms = 100ms * 15回, global_countが15回以下ならば
//global_countとkaisuuを0にする
      global_count_click = 0;
      kaisuu_click = 0;
}

      msg_click[0] = 0;//msg変数を0と置く.変数の初期化
```

```
421 if (val_click >= Shikiiti){
422 //しきい値 ： 測定電圧1.21065（Ｖ）÷基準電圧(電源電圧)（５Ｖ）
423 //Ｘ 1023(変換段階数) ＝ A/D変換出力値（しきい値）247.7
424 //しきい値が2回以上超える
425       if (kaisuu_click < 2) {
426         kaisuu_click = kaisuu_click + 1;
427       } else { //  しきい値より大きくなった&2回以上なら以下を処理
428         msg_click[0] = 1;//しきい値を超えたらmsg変数を1にする
429 
430         kaisuu_click = 0;
431         global_count_click = 0;
432       }
433     }
434 
435 /*
436 息圧判定により息と判定されスイッチがONのとき、かつ前回は息がはいってない
437 （スイッチがOFF）のとき 、タブレットに送信
438 */
439 if(msg_click[0] != msg_old_click && msg_click[0]==1){
440 /*
441 msgとmsg_click(previous)が等しくない(!=)、
442 かつprevious0⇒1（msg[0] != previousではprevious0⇒1とprevious1⇒0、
443 すなわち、しきい値が超えたときと戻る時を検出してしまうので
444 msg[0]==1の時のみ以下を行う）ならば（時刻nのとき
445 スイッチがON、前回がOFFだったとき
446 */
447 
448 //LED点滅の状態を切り替える
449       if(G_state_click == FLG_ON){      //  フラグがONのとき
450         G_state_click = FLG_OFF;             // OFF
451        } else {                          //  フラグがOFFのとき
452         G_state_click = FLG_ON;              // ON
453        }
454   }
455 
456 msg_old_click = msg_click[0];                //  スイッチの状態をG_oldval_clickへ保存
457 
458 //LEDの点滅状態を開始するとき
459 if (G_state_click == FLG_ON){           // LEDの点滅状態がONのとき
460 
461       digitalWrite(LED_click, SW_ON);       //  緑色LED点灯
462       Mouse.click();//右クリックを送信する
463 
464       delay(100);
465       digitalWrite(LED_click, SW_OFF);
466 
467 
468       //他のスイッチを切る
469         digitalWrite(LED_up, SW_OFF);
470         digitalWrite(LED_down, SW_OFF);
471         digitalWrite(LED_left, SW_OFF);
472         digitalWrite(LED_right, SW_OFF);
473 
```

```
474
475         //フラグをリセット
476           G_state_up = FLG_OFF;
477           G_state_down = FLG_OFF;
478           G_state_left = FLG_OFF;
479           G_state_right = FLG_OFF;
480           G_state_click = FLG_OFF;
481
482           global_count_click = global_count_click + 1;//global_count_rightを1増やす
483
484   } else   { // OFFのとき
485           digitalWrite( LED_click, SW_OFF ); // LEDを消灯
486   }
487   delay(10);                          // チャタリング防止のウエイト
488   }
489
```

II　Eclipse のプログラム（Java）

ii－1 Android タブレットのスリープ解除プログラム

/DigitalIIn/src/aoabook/sample/chap3/digitalin/MainActivity.java

```
1   package aoabook.sample.chap3.digitalin;
2
3   import java.io.FileDescriptor;
4   import java.io.FileInputStream;
5   import java.io.IOException;
6
7   import android.app.Activity;
8   import android.app.AlarmManager;
9   import android.app.PendingIntent;
10  import android.content.BroadcastReceiver;
11  import android.content.Context;
12  import android.content.Intent;
13  import android.content.IntentFilter;
14  import android.graphics.Color;
15  import android.os.Bundle;
16  import android.os.ParcelFileDescriptor;
17  import android.util.Log;
18  import android.widget.TextView;
19  import android.widget.Toast;
20
21
22  import android.hardware.usb.UsbAccessory;
23  import android.hardware.usb.UsbManager;
24
25
26  /*
27   * Android,Arduino間で通信を行う
28   */
29  public class MainActivity extends Activity {
30   private static final String TAG = "DigitalIn";
31
32   private static final String ACTION_USB_PERMISSION =
33           "aoabook.sample.accessory.action.USB_PERMISSION";
34
35   private TextView mStatusText;
36
37   private UsbManager mUsbManager;
38   private PendingIntent mPermissionIntent;
39   private boolean mPermissionRequestPending;
40
41   private UsbAccessory mAccessory;
42   private ParcelFileDescriptor mFileDescriptor;
```

```
 private FileInputStream mInputStream;

 private boolean mThreadRunning = false;



//USB接続状態変化のインテントを受け取るレシーバ

private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver()  {
 @Override
 public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
   if (ACTION_USB_PERMISSION.equals(action)) {
    Toast.makeText(MainActivity.this, "receiver",
    Toast.LENGTH_SHORT).show();

//ユーザーが確認ダイアログでOKまたはキャンセルを押した場合
     synchronized (this) {
     //UsbAccsessoryインスタンスを取得
UsbAccessory accessory = (UsbAccessory)
intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
     //ユーザーがアクセスを許可した場合
      if (intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED,
          false)) {
     //アクセサリをオープン
          openAccessory(accessory);
       }else {
          Log.d(TAG, "permission denied for accessory " + accessory);
       }
          mPermissionRequestPending =false;
}
} else if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {

//アクセサリが切断された場合
UsbAccessory accessory = (UsbAccessory)
intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);

if (accessory != null && accessory.equals(mAccessory)) {
//アクセサリをクローズ
    closeAccessory();
                                  }
                        }
          }
};


@Override
public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);

          mStatusText = (TextView)findViewById(R.id.text_status);
          //UsbManagerのインスタンスを取得
```

```
96              mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
97
98              //ユーザーからアクセス拒否のアクションがあったときに
99              //ブロードキャストされるインテント
100             mPermissionIntent = PendingIntent.getBroadcast(this, 0,
101             new Intent(ACTION_USB_PERMISSION), 0);
102
103             //フィルタを設定し、ブロードキャストレシーバーを登録
104             IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
105             filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
106             registerReceiver(mUsbReceiver, filter);
107 }
108
109
110 @Override
111 public void onResume() {
112 super.onResume();
113
114 //アクセサリリストを取得
115 UsbAccessory[] accessories = mUsbManager.getAccessoryList();
116 //現状の仕様ではアクセサリは1つなので先頭要素のみ
117 UsbAccessory accessory = (accessories == null ? null : accessories[0]);
118 if(accessory != null) {
119   if (mUsbManager.hasPermission(accessory)) {
120     //パーミッションを取得している場合はアクセサリをオープン
121     openAccessory(accessory);
122     } else {
123     //パーミッションを取得していない場合は
124     //requestPermission()を呼び出してユーザーにパーミッションを要求
125     synchronized (mUsbReceiver) {
126     if (!mPermissionRequestPending) {
127
128 //アクセス許可ダイアログを表示
129 mUsbManager.requestPermission(accessory, mPermissionIntent);
130 mPermissionRequestPending = true;
131                                                 }
132                                             }
133                                         }
134     } else {
135 Log.d(TAG, "mAccessory is null");
136                         }
137             }
138
139
140 @Override
141 public void onPause() {
142 super.onPause();
143 //アクセサリをクローズ
144 closeAccessory();
145 }
146
147
148 @Override
```

```
149 public void onDestroy() {
150       //ブロードキャストレシーバーの登録解除
151       unregisterReceiver(mUsbReceiver);
152       super.onDestroy();
153 }
154
155
156       //アクセサリをオープン
157       private void openAccessory(UsbAccessory accessory) {
158             mFileDescriptor = mUsbManager.openAccessory(accessory);
159             if (mFileDescriptor != null) {
160                FileDescriptor fd = mFileDescriptor.getFileDescriptor();
161                mInputStream = new FileInputStream(fd);
162
163                new Thread(new MyRunnable()).start();
164             } else {
165             Log.d(TAG, "Faild to open the accessory");
166             }
167       }
168
169 //アクセサリをクローズ
170 private void closeAccessory() {
171    mThreadRunning = false;
172    try {
173    if (mFileDescriptor != null) {
174     mFileDescriptor.close();
175     }
176          } catch (IOException e) {
177             e.printStackTrace();
178          } finally {
179             mInputStream = null;
180             mFileDescriptor = null;
181             mAccessory = null;
182           }
183 }
184
185 class MyRunnable implements Runnable {
186  @Override
187  public void run() {
188    mThreadRunning = true;
189    while (mThreadRunning) {
190             byte[] buffer = new byte[1];
191             try {
192             //入力スロリームの読み込み
193             mInputStream.read(buffer);
194
195             //UI処理はメインスレッドで行う
196             if (buffer[0] == 1) {
197                Intent MyIntent = new Intent("ScreenLockManagerTag.VIEW");
198                sendBroadcast(MyIntent);
199
200
201             MainActivity.this.runOnUiThread(new Runnable() {
```

```
202                 @Override
203                 public void run() {
204                   mStatusText.setText("Status ON");
205                   mStatusText.setBackgroundColor(Color.RED);
206                                       }
207               });
208               } else {
209
210               MainActivity.this.runOnUiThread(new Runnable() {
211                   @Override
212                   public void run() {
213                     mStatusText.setText("Status OFF");
214                     mStatusText.setBackgroundColor(Color.BLACK);
215                   }
216               });
217               }
218               } catch (IOException e) {
219                 e.printStackTrace();
220               }
221               }
222 };}}
223
224
```

/DigitalIIn/src/aoabook/sample/chap3/digitalin/ScreenLockManager2kidouzi.java

```
1   package aoabook.sample.chap3.digitalin;
2
3   import android.app.KeyguardManager;
4   import android.content.BroadcastReceiver;
5   import android.content.Context;
6   import android.content.Intent;
7   import android.os.PowerManager;
8   import android.util.Log;
9   import android.app.KeyguardManager.KeyguardLock;
10
11  public class ScreenLockManager2kidouzi extends BroadcastReceiver {
12
13  private static final String KEYGUARD_SERVICE = null;
14  private static PowerManager.WakeLock wakelock;
15
16
17  @SuppressWarnings("deprecation")
18  @Override
19  public void onReceive(Context context, Intent intent){
20     Log.d("TEST","called");
21
22     //スマホの電源確保、スリープ状態から復帰する
23     wakelock = ((PowerManager)
24     context.getSystemService(Context.POWER_SERVICE))
25     .newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK
```

```
26      | PowerManager.ACQUIRE_CAUSES_WAKEUP
27      | PowerManager.ON_AFTER_RELEASE, "disableLock");
28      wakelock.acquire();
29
30      //スクリーンロックを解除する
31      KeyguardManager km = (KeyguardManager)
32      context.getSystemService(Context.KEYGUARD_SERVICE);
33      KeyguardManager.KeyguardLock klock = km.newKeyguardLock("disableLock");
34      klock.disableKeyguard();
35
36
37       //アラームを実行するMainActivity.javaへ
38       Intent intent1 = new Intent(context, MainActivity.class);
39       intent1.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
40       context.startActivity(intent1);
41  }
42
43
44  //アプリが終わったらrelease関数を呼ぶ（ずっと電源がつきっぱなしになってしまう為）
45  public static void release(){
46      if(wakelock != null){
47        wakelock.release();
48      }
49  }
50       public static String getKeyguardService() {
51       return KEYGUARD_SERVICE;
52       }
53  }
54
```

/DigitalIIn/src/aoabook/sample/chap3/digitalin/StartupReceiver.java

```
1   package aoabook.sample.chap3.digitalin;
2
3   import android.content.BroadcastReceiver;
4   import android.content.Context;
5   import android.content.Intent;
6
7   public class StartupReceiver extends BroadcastReceiver{
8    @Override
9    public void onReceive(Context context, Intent intent) {
10      Intent serviceIntent = new Intent(context, ScreenLockManager2kidouzi.class);
11      serviceIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
12      context.startActivity(serviceIntent);
13    }
14  }
```

/DigitalIIn/src/aoabook/sample/chap3/digitalin/ScreenLockManager.java

```
1   package aoabook.sample.chap3.digitalin;
2
3   import android.content.BroadcastReceiver;
4   import android.app.KeyguardManager;
5   import android.app.KeyguardManager.KeyguardLock;
6   import android.content.Context;
7   import android.content.Intent;
8   import android.os.PowerManager;
9   import android.util.Log;
10
11
12  public class ScreenLockManager extends BroadcastReceiver {
13     private static final String KEYGUARD_SERVICE = null;
14     private static PowerManager.WakeLock wakelock;
15
16     @SuppressWarnings("deprecation")
17     @Override
18     public void onReceive(Context context, Intent intent){
19       Log.d("TEST","called");
20
21       //スマホの電源確保、スリープ状態から復帰する
22     wakelock = ((PowerManager)
23     context.getSystemService(Context.POWER_SERVICE))
24                          .newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK
25       | PowerManager.ACQUIRE_CAUSES_WAKEUP
26       | PowerManager.ON_AFTER_RELEASE, "disableLock");
27       wakelock.acquire();
28
29
30       //スクリーンロックを解除する
31       KeyguardManager km = (KeyguardManager)
32       context.getSystemService(Context.KEYGUARD_SERVICE);
33       KeyguardManager.KeyguardLock klock = km.newKeyguardLock("disableLock");
34       klock.disableKeyguard();
35     }
36
37  //アプリが終わったらrelease関数を呼ぶ事
38  //（ずっと電源がつきっぱなしになってしまう為）
39     public static void release(){
40        if(wakelock != null){
41              wakelock.release();
42        }
43     }
44
45     public static String getKeyguardService() {
46        return KEYGUARD_SERVICE;
47      }
48  }
```

/DigitalIIn/AndroidManifest.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3         package="aoabook.sample.chap3.digitalin"
4         android:versionCode="1"
5         android:versionName="1.0" >
6
7   <uses-sdk
8         android:minSdkVersion="13"
9         android:targetSdkVersion="17" />
10        <uses-permission android:name="android.permission.WAKE_LOCK"/>
11        <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
12        <uses-permission
13        android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
14  <uses-feature />
15
16  <application
17        android:allowBackup="true"
18        android:icon="@drawable/ic_launcher"
19        android:label="@string/app_name"
20        android:theme="@style/AppTheme" >
21        <activity
22              android:name="aoabook.sample.chap3.digitalin.MainActivity"
23              android:label="@string/app_name" >
24              <intent-filter>
25                    <action android:name="android.intent.action.MAIN" />
26                    <category android:name="android.intent.category.LAUNCHER" />
27              </intent-filter>
28              <intent-filter>
29       <action
30       android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
31               </intent-filter>
32  <meta-data
33        android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
34        android:resource="@xml/accessory_filter" />
35        </activity>
36
37  <receiver
38        android:name=".StartupReceiver"
39        android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
40                <intent-filter>
41                <action android:name="android.intent.action.BOOT_COMPLETED" />
42                <category android:name="android.intent.category.DEFAULT" />
43                </intent-filter>
44  </receiver>
45
46  <receiver
47        android:name="aoabook.sample.chap3.digitalin.ScreenLockManager2kidouzi"
48        android:exported="false" >
49  <intent-filter>
50  <action android:name="ScreenLockManager2kidouziTag.VIEW"    />
51  </intent-filter>
```

```
52  </receiver>
53
54  <receiver
55        android:name="aoabook.sample.chap3.digitalin.ScreenLockManager"
56        android:exported="false" >
57  <intent-filter>
58  <action android:name="ScreenLockManagerTag.VIEW"    />
59  </intent-filter>
60  </receiver>
61  </application>
62  </manifest>
```

/DigitalIIn/gen/aoabook/sample/chap3/digitalin/R.java

```
1   /* AUTO-GENERATED FILE.    DO NOT MODIFY.
2    *
3    * This class was automatically generated by the
4    * aapt tool from the resource data it found.    It
5    * should not be modified by hand.
6    */
7
8   package aoabook.sample.chap3.digitalin;
9
10  public final class R {
11        public static final class attr {
12        }
13        public static final class dimen {
14              /**    Default screen margins, per the Android Design guidelines.
15
16                Customize dimensions originally defined in res/values/dimens.xml (such as
17                screen margins) for sw720dp devices (e.g. 10" tablets) in landscape here.
18
19              */
20          public static final int activity_horizontal_margin=0x7f050000;
21          public static final int activity_vertical_margin=0x7f050001;
22          }
23        public static final class drawable {
24              public static final int ic_launcher=0x7f020000;
25        }
26        public static final class id {
27              public static final int action_settings=0x7f090001;
28              public static final int text_status=0x7f090000;
29        }
30        public static final class layout {
31              public static final int activity_main=0x7f030000;
32        }
33        public static final class menu {
34              public static final int main=0x7f080000;
35        }
36        public static final class string {
37              public static final int action_settings=0x7f060001;
```

```
38              public static final int app_name=0x7f060000;
39              public static final int hello_world=0x7f060002;
40          }
41          public static final class style {
42              /**
43              Base application theme, dependent on API level. This theme is replaced
44              by AppBaseTheme from res/values-vXX/styles.xml on newer devices.
45
46                  Theme customizations available in newer API levels can go in
47                  res/values-vXX/styles.xml, while customizations related to
48                  backward-compatibility can go here.
49
50              Base application theme for API 11+. This theme completely replaces
51              AppBaseTheme from res/values/styles.xml on API 11+ devices.
52
53      API 11 theme customizations can go here.
54               */
55              public static final int AppBaseTheme=0x7f070000;
56              /**    Application theme.
57      All customizations that are NOT specific to a particular API-level can go here.
58               */
59              public static final int AppTheme=0x7f070001;
60          }
61          public static final class xml {
62              public static final int accessory_filter=0x7f040000;
63          }
64      }
```

Ⅲ　Microsoft Visual C++ 2008 Express Edition のプログラム（C++）

ⅲ－1 Windows タブレットの顔の移動と位置によるマウス操作のプログラム

オプティカルフロー.cpp (ソースファイル)

```
1   // オプティカルフロー.cpp : コンソールアプリケーションのエントリポイントを定義しま
2   す。
3   //
4
5
6   #include "stdafx.h"
7   #include <cv.h>
8   #include <highgui.h>
9   #include <windows.h>
10
11  #pragma comment(lib,"cv.lib")
12  #pragma comment(lib,"cxcore.lib")
13  #pragma comment(lib,"highgui.lib")
14
15  #include <time.h>           // 時間関連ヘッダ
16  #include <math.h>           // 数学関連ヘッダ
17
18  /*  グローバル変数*/
19  CvFont font;
20  IplImage *frame=0;                                  // 画像変数宣言
21  IplImage *img = 0;
22
23  /*  プロトタイプ宣言*/
24  void on_mouse (int event, int x, int y, int flags, void *param);
25
26  int _tmain(int argc, _TCHAR* argv[])
27  {
28          CvSeq *objects;
29          CvSeq *objects2;
30
31          CvRect *r;
32          CvRect lower_half;
33
34          char *cascade_name;                 // カスケードファイル名
35
36          int i, key, count = 500;                        // 特徴点の個数
37          char* status = new char[count];
38
```

```
39          CvPoint2D32f *feature_pre, *feature_now; // 浮動小数点数型座標の特徴点
40          feature_pre = new CvPoint2D32f[count];
41          feature_now = new CvPoint2D32f[count];
42
43          CvCapture* src;                         // ビデオキャプチャ宣言
44
45
46          IplImage *frame_copy=0;                 // フレーム単位データコピー用
47          IplImage *gray=0;
48          IplImage *detect_frame=0;
49
50          // 画像変数宣言
51          IplImage *frame_pre=0;
52          IplImage *frame_now=0;
53
54          // 画像変数宣言
55          IplImage *frame_pre_face=0;
56          IplImage *img_out=0;
57          IplImage *img_tmp1=0;
58          IplImage *img_tmp2=0;
59          IplImage *pyramid_now=0;
60          IplImage *pyramid_pre=0;
61
62          // 反復アルゴリズム用終了条件
63          CvTermCriteria criteria;
64          criteria = cvTermCriteria(CV_TERMCRIT_ITER
65          | CV_TERMCRIT_EPS, 20, 0.03);
66
67          CvHaarClassifierCascade *cascade;       // カスケード分類器の宣言
68          CvMemStorage *storage;                  // メモリ領域の宣言
69
70          CvHaarClassifierCascade *cascade2;      // カスケード分類器の宣言
71          CvMemStorage *storage2;                 // メモリ領域の宣言
72
73
74          // カスケード・ファイルの読み込み
75          cascade_name = "haarcascade_frontalface_alt_tree.xml";
76          cascade = (CvHaarClassifierCascade *) cvLoad(cascade_name, 0, 0, 0);
77          storage = cvCreateMemStorage (0);       // メモリ領域の確保
78          cvClearMemStorage(storage);             // メモリ領域の初期化
79
80  /*
81          // カスケード・ファイルの読み込み
82          cascade_name = "haarcascade_mcs_mouth.xml";
83          cascade2 = (CvHaarClassifierCascade *) cvLoad(cascade_name, 0, 0, 0);
84          storage2 = cvCreateMemStorage (0);      // メモリ領域の確保
85          cvClearMemStorage(storage2);            // メモリ領域の初期化
86  */
87  /*
88  カメラの選択
89  Windowsタブレッド
90  0=外カメラ(rear camera)、1=内カメラ(front camera)、2=USBカメラ（外付け）
91
```

```
92
93  PC（PCにカメラがついてない場合）
94  0=USBカメラ（外付け）
95  */
96          src = cvCaptureFromCAM(0);              // 映像取得（カメラ映像）
97
98          //カメラから取得した画像のサイズ指定
99          cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_WIDTH, 640);
100         cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_HEIGHT, 480);
101
102         if(src == NULL){printf("映像が取得できません。¥n"); cvWaitKey(0); return -1;}
103         cvNamedWindow ("顔検出");
104
105         int STOPPING = 0;
106         int MOVING_LEFT = 1;
107         int MOVING_RIGHT = 2;
108         int MOVING_UP = 3;
109         int MOVING_DOWN = 4;
110
111         int movingState = STOPPING;
112
113         int BORDER_HORIZONTAL_1 = 5;
114         int BORDER_HORIZONTAL_2 = 10;
115
116         int BORDER_VERTICAL_1 = 2;
117         int BORDER_VERTICAL_2 = 4;
118
119         int stopping_count = 0;
120         int right_count = 0;
121         int left_count = 0;
122         int up_count = 0;
123         int down_count = 0;
124         int COUNT_MAX = 5;
125
126         //繰り返しカウント(時間)
127         int count_click = 0;
128
129         while(1){
130          frame = cvQueryFrame(src); if(frame == NULL) break; // 1フレーム取得
131
132         if(frame_pre==0) frame_pre = cvCreateImage
133             (cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1);
134         if(frame_now==0) frame_now = cvCreateImage
135             (cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1);
136
137         if(img_out==0) img_out
138        = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U, 3);
139         if(img_tmp1==0) img_tmp1
140        = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_32F, 1);
141         if(img_tmp2==0) img_tmp2
142        = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_32F, 1);
143         if(pyramid_pre==0) pyramid_pre
144         = cvCreateImage(cvSize(frame->width+8, frame->height/3),IPL_DEPTH_8U,
```

```
145             1);
146             if(pyramid_now==0) pyramid_now
147             = cvCreateImage(cvSize(frame->width+8, frame->height/3),IPL_DEPTH_8U,
148             1);
149
150             // フレームコピー用イメージ生成
151             if(frame_copy==0) frame_copy = cvCreateImage
152             (cvSize(frame->width, frame->height), IPL_DEPTH_8U, frame->nChannels);
153             if(frame->origin == IPL_ORIGIN_TL) {
154                 cvCopy(frame, frame_copy);
155             } else {
156                 cvFlip(frame, frame_copy);
157             }
158
159             cvCvtColor(frame, frame_now, CV_BGR2GRAY); // グレースケール変換
160
161             if(frame_pre_face==0){
162                 frame_pre_face
163                 = cvCreateImage(cvSize(frame->width, frame->height), IPL_DEPTH_8U,
164                 1);
165                 cvCopy(frame_now, frame_pre_face);
166                 //はじめの１回はframe_pre_faceが無いので
167                 //frame_nowをframe_pre_faceにする為にコピーする
168             }
169
170             // 読み込んだ画像のグレースケール化、及びヒストグラムの均一化を行う
171             if(gray==0) gray = cvCreateImage
172                 (cvSize(frame_copy->width, frame_copy->height), IPL_DEPTH_8U, 1);
173             if(detect_frame==0) detect_frame = cvCreateImage
174                 (cvSize((frame_copy->width), (frame_copy->height)), IPL_DEPTH_8U, 1);
175             cvResetImageROI(detect_frame);
176             cvCvtColor(frame_copy, gray, CV_BGR2GRAY);
177             cvResize(gray, detect_frame, CV_INTER_LINEAR);
178             cvEqualizeHist(detect_frame, detect_frame);
179
180
181             //移動範囲の視覚化
182             //原点を移動する
183             //上に移動の際、顔が切れて認識しなくなるため
184             int genten_idou_x = 240;
185             int genten_idou_y = 160;
186
187             //切り替え範囲x、y
188             int x0 = 0    + genten_idou_x;
189             int x1 = 57 + genten_idou_x;
190
191             int x2 = 103 + genten_idou_x;
192             int x3 = 160 + genten_idou_x;
193
194             int y0 = 0    + genten_idou_y;
195             int y1 = 45   + genten_idou_y;
196             int y2 = 85 + genten_idou_y;
197             int y3 = 115 + genten_idou_y;
```

```
198             int y4 = 120 + genten_idou_y;
199             int y5 = 160 + genten_idou_y;
200
201             int l1x = (x2 + 25);
202             int r1x = (x1-25) ;
203
204             //矢印の変数
205             int saunkaku_x = 10;
206             int saunkaku_y = 14 /2;
207             int saunkaku_h = 5;
208
209             //顔検出のフレームの中心座標
210             int tyuushin_x = 80+ genten_idou_x;
211             int tyuushin_y = 65+ genten_idou_y;
212
213             //範囲を灰色で表示（x,y）
214             cvLine(frame, cvPoint(x1,y1), cvPoint(x1,y4), CV_RGB (200, 200, 203),
215             4, CV_AA, 0); //T1
216             cvLine(frame, cvPoint(x2,y1), cvPoint(x2,y4), CV_RGB (200, 200, 203),
217             4, CV_AA, 0); //T2
218             cvLine(frame, cvPoint(x0,y1), cvPoint(x3,y1), CV_RGB (200, 200, 203),
219             4, CV_AA, 0); //Y1
220             cvLine(frame, cvPoint(x1,y2), cvPoint(x2,y2), CV_RGB (200, 200, 203),
221             4, CV_AA, 0); //Y1-2
222             cvLine(frame, cvPoint(x0,y3), cvPoint(x1,y3), CV_RGB (200, 200, 203),
223             4, CV_AA, 0); //Y2a
224             cvLine(frame, cvPoint(x2,y3), cvPoint(x3,y3), CV_RGB (200, 200, 203),
225             4, CV_AA, 0); //Y2b
226             cvLine(frame, cvPoint(x1,y4), cvPoint(x2,y4), CV_RGB (200, 200, 203),
227             4, CV_AA, 0); //Y2-3
228
229 //cvLine(frame, cvPoint(r1x,y1), cvPoint(r1x,y3), CV_RGB (255, 0, 0), 4, CV_AA, 0);
230 //クリックとの境目　y_c1
231 //cvLine(frame, cvPoint(x0,y1), cvPoint(x0,y3), CV_RGB (0, 255, 0), 4, CV_AA, 0);
232 //クリックとの境目　y_c1
233 //cvLine(frame, cvPoint(x1,y1), cvPoint(x1,y3), CV_RGB (0, 0, 255), 4, CV_AA, 0);
234 //クリックとの境目　y_c1
235
236             //ただし、反転注意（右三角を描くと左三角を表示）
237             //停止
238             //中心よりやや上に灰色の○を描画
239             //赤色CV_RGB CV_RGB(255, 40, 0)
240             //口は下の方にあるため
241             //首の可動域は左右よりも上下の方が小さい
242             cvCircle(frame /* 描画するフレームの指定*/, cvPoint(tyuushin_x, tyuushin_y)
243             /* 円の中心（顔の中心）座標*/,14/* 円の大きさ*/, CV_RGB(200, 200, 203)
244             /* 色*/, 2 /* 線の太さ*/, CV_AA, 0);
245
246             //右矢印｜→（右向き三角と線）を灰色で描画
247             //ピンク色CV_RGB (255, 153, 160)
248             //三角
249             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
250
```

```
251             cvPoint(32 + genten_idou_x ,80 + genten_idou_y), CV_RGB (200, 200, 203),
252             12, CV_AA, 0); //
253             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
254             cvPoint(32 + genten_idou_x ,90+ genten_idou_y), CV_RGB (200, 200, 203),
255             12, CV_AA, 0); // |
256             cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
257             cvPoint(32+ genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
258             12, CV_AA, 0); //／
259
260             //左矢印← |（左向き三角と線）を灰色で描画
261             ////空色CV_RGB (102, 204, 255)
262              cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
263              cvPoint(128 + genten_idou_x ,80 + genten_idou_y), CV_RGB (200, 200, 203),
264              12, CV_AA, 0); //
265              cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
266              cvPoint(128 + genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
267              12, CV_AA, 0); //__
268              cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
269              cvPoint(128 + genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
270              12, CV_AA, 0); //／
271
272             //上矢印__↑（上向き三角と線）を灰色で描画
273             //茶色CV_RGB (102, 51, 0)
274             cvLine(frame, cvPoint(80 + x0, 17 + y0), cvPoint(87 + x0 ,27+ y0),
275             CV_RGB (200, 200, 203), 12, CV_AA, 0); //A
276             cvLine(frame, cvPoint(73 + x0, 27+ y0), cvPoint(80+ x0 ,17 + y0),
277             CV_RGB (200, 200, 203), 12, CV_AA, 0); //B
278             cvLine(frame, cvPoint(87 + x0,27 + y0), cvPoint(73 + x0 ,27 + y0),
279             CV_RGB (200, 200, 203), 12, CV_AA, 0); //C
280
281             //下矢印__↓（下向き三角と線）を灰色で描画
282             //黄色CV_RGB (255, 245, 0)
283             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
284             cvPoint(87 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
285             12, CV_AA, 0); //／
286             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
287             cvPoint(73 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
288             12, CV_AA, 0); //__
289             cvLine(frame, cvPoint(87 + genten_idou_x,133 + genten_idou_y),
290             cvPoint(73 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
291             12, CV_AA, 0); //
292
293
294
295             // Haar-Like顔検出
296             objects = cvHaarDetectObjects(detect_frame, cascade, storage, 1.1, 3, 0,
297             cvSize(0, 0));
298
299             int face_max_size = 0;
300             int face_max_index = -1;
301             CvRect *r_face_max=0;
302
303             // 検出数の繰り返し
```

```
304             for (int i = 0; i < (objects ? objects->total : 0); i++) {
305               // 検出矩形領域の取得rは顔の描画の起点座標
306               r = (CvRect *) cvGetSeqElem(objects, i);
307
308             //一番大きな顔を使用
309             int length = r->width + r->height;
310
311                if(length > face_max_size){
312                   face_max_size = length;
313                   r_face_max = r;
314                   face_max_index = i;
315                }
316              }
317
318             if(r_face_max!=0){
319
320             /*
321             cvRectangle(frame, cvPoint(r_face_max->x, r_face_max->y)
322             // 検出部に四角枠描画
323             , cvPoint(r_face_max->x + r_face_max->width, r_face_max->y +
324             r_face_max->height), CV_RGB(255, 0, 0), 4); //赤
325             */
326
327                //顔の(x,y)=(1/2, 3/4)に赤色の●を描画
328                cvCircle(frame /* 描画するフレームの指定*/,
329                cvPoint(r_face_max->x + r_face_max->width/2, r_face_max->y + 3
330                *(r_face_max->height)/4) /* 円の中心（顔の中心）座標*/,
331                2/* 円の大きさ*/, CV_RGB(255, 40, 0) /* 色*/, 16 /* 線の太さ*/, CV_AA, 0);
332
333             // この時点で、最大の顔のオブジェクトの矩形領域はr_face_max
334             //顔の下半分の領域
335             lower_half = cvRect(r_face_max->x + r_face_max->width / 3, r_face_max->y +
336             2*r_face_max->height / 3, //原点から/3幅x,2/3の高さyをずらす①
337             r_face_max->width/3, r_face_max->height / 3); //縮尺
338
339             cvResetImageROI(detect_frame);
340             cvSetImageROI(detect_frame, lower_half);
341 /*
342             // Haar-Like口検出
343             objects2 = cvHaarDetectObjects(detect_frame, cascade2, storage2,
344               1.1, 3, 0, cvSize(0, 0));
345
346             // 検出数の繰り返し
347             for (int i = 0; i < (objects2 ? objects2->total : 0); i++) {
348             r = (CvRect *) cvGetSeqElem(objects2, i);   // 検出矩形領域の取得
349             cvRectangle(frame, cvPoint( lower_half.x + r->x, lower_half.y + r->y)
350             // 検出部に四角枠描画
351             , cvPoint(lower_half.x +r->x + r->width, lower_half.y +r->y + r->height),
352             CV_RGB(0, 255, 0), 4); //緑
353             }
354 */
355 /*
356
```

```
357             r_face_max2 = cvRect(r_face_max->x + r_face_max->width / 3,
358             r_face_max->y +2*r_face_max->height / 3, //原点から/3幅x,
359             2/3の高さyをずらす①r_face_max->width/3, r_face_max->height / 3); //縮尺
360
361             cvResetImageROI(frame_pre);
362             cvSetImageROI(frame_pre, r_face_max2);
363     */
364
365             // pre_faceからの特徴点の抽出
366             cvGoodFeaturesToTrack(frame_pre_face, img_tmp1, img_tmp2,
367             feature_pre, &count, 0.001, 5, NULL);
368
369             // オプティカルフロー検出
370             cvCalcOpticalFlowPyrLK(frame_pre_face, frame_now, pyramid_pre,
371             pyramid_now,feature_pre, feature_now, count, cvSize(10, 10),
372             4, status, NULL, criteria, 0);
373
374             //オプティカルフローif文ここから
375             int ave_x = 0;
376             int ave_y = 0;
377             int ave_count = 0;
378             for(i = 0; i < count; i++){                 // オプティカルフロー描画
379                 double x_abs = fabs((double)(cvPointFrom32f(feature_now[i]).x
380                  - cvPointFrom32f(feature_pre[i]).x)); // ベクトルx成分の絶対値
381                 double y_abs = fabs((double)(cvPointFrom32f(feature_now[i]).y
382                  - cvPointFrom32f(feature_pre[i]).y)); // ベクトルy成分の絶対値
383
384                 if(x_abs < 640/4 && y_abs < 480/4){//    「&&」は条件A「かつ」B
385                     ave_x += cvPointFrom32f(feature_now[i]).x
386                      - cvPointFrom32f(feature_pre[i]).x;
387                     ave_y += cvPointFrom32f(feature_now[i]).y
388                      - cvPointFrom32f(feature_pre[i]).y;
389                     ave_count++;
390                 }
391             }
392
393             ave_x *= 10;
394             ave_y *= 10;
395
396             if(ave_count != 0){ //1個でも点があったら
397                 ave_x /= ave_count;
398                 ave_y /= ave_count;
399             }
400
401             POINT po;//カーソル位置用
402             GetCursorPos(&po);//マウスの座標を取得
403
404             // 顔の●の定義
405             int mouth_x = (r_face_max->x + r_face_max->width/2);
406             int mouth_y = (r_face_max->y + 3 *(r_face_max->height)/4);
407
408             //左←←
409
```

```
410             if(x2<mouth_x && mouth_x<l1x && y1<mouth_y && mouth_y<y3){
411                 po.x -= 30;
412                 SetCursorPos(po.x ,po.y);
413
414                 //左矢印←｜（左向き三角と線）を空色で描画
415                 //空色CV_RGB (102, 204, 255)
416                 //三角
417                 cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
418                 cvPoint(128 + genten_idou_x ,80 + genten_idou_y),
419                 CV_RGB (102, 204, 255),
420                 12, CV_AA, 0); //
421                 cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
422                 cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
423                 CV_RGB (102, 204, 255),
424                 12, CV_AA, 0); //__
425                 cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
426                 cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
427                 CV_RGB (102, 204, 255),
428                 12, CV_AA, 0); //／
429
430                 count_click = 0;
431             }
432
433             //左←
434             else if(l1x<mouth_x && mouth_x<x3 && y1<mouth_y && mouth_y<y3){
435
436              //向きA　左空色
437              //空色CV_RGB (102, 204, 255)
438              if(ave_y < 3*ave_x /4 && ave_y >= -3*ave_x /4){
439                  po.x -= 30;
440                  SetCursorPos(po.x ,po.y);
441
442                 //左矢印←（左向き三角）を空色で描画
443                 //空色CV_RGB (102, 204, 255)
444                  cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
445                  cvPoint(128 + genten_idou_x ,80 + genten_idou_y),
446                  CV_RGB (102, 204, 255),
447                  12, CV_AA, 0); //
448                  cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
449                  cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
450                  CV_RGB (102, 204, 255),
451                  12, CV_AA, 0); //__
452                  cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
453                  cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
454                  CV_RGB (102, 204, 255),
455                  12, CV_AA, 0); //／
456              }
457
458                 count_click = 0;
459             }
460
461
462
```

```
463         //右→→
464         else if(x0<mouth_x && mouth_x<r1x && y1<mouth_y && mouth_y<y3){
465                 po.x += 30;
466                 SetCursorPos(po.x ,po.y);
467
468                  //右矢印｜→（右向き三角と線）をピンク色で描画
469                  //ピンク色CV_RGB (255, 153, 160)
470                  //三角
471                 cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
472                 cvPoint(32 + genten_idou_x ,80 + genten_idou_y),
473                 CV_RGB(255, 153, 160) ,
474                 12, CV_AA, 0); //
475                 cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
476                 cvPoint(32 + genten_idou_x ,90+ genten_idou_y),
477                 CV_RGB (255, 153, 160) ,
478                 12, CV_AA, 0); // |
479                 cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
480                 cvPoint(32+ genten_idou_x ,90 + genten_idou_y),
481                 CV_RGB (255, 153, 160) ,
482                 12, CV_AA, 0); //／
483
484                 count_click = 0;
485         }
486
487
488         //右→
489         else if (r1x<mouth_x && mouth_x<x1 && y1<mouth_y && mouth_y<y3){
490
491          //向きC　右→ピンク
492          //ピンク色CV_RGB (255, 153, 160)
493           if(ave_y <= -3*ave_x /4 && ave_y > 3*ave_x /4){
494                   po.x += 30;
495                   SetCursorPos(po.x ,po.y);
496
497                   //右矢印→（右向き三角）をピンク色で描画
498                   //ピンク色CV_RGB (255, 153, 160)
499                   cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
500                   cvPoint(32 + genten_idou_x ,80 + genten_idou_y),
501                 CV_RGB(255, 153, 160) , 12, CV_AA, 0); //
502                 cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
503                   cvPoint(32 + genten_idou_x ,90+ genten_idou_y),
504                   CV_RGB (255, 153, 160) , 12, CV_AA, 0); // |
505                   cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
506                   cvPoint(32+ genten_idou_x ,90 + genten_idou_y),
507                   CV_RGB (255, 153, 160) , 12, CV_AA, 0); //／
508          }
509            count_click = 0;
510         }
511
512
513         //上↑↑
514         else if(x0<mouth_x && mouth_x<x3 && y0<mouth_y && mouth_y<y1){
515
```

```
516                 po.y -= 30;
517                 SetCursorPos(po.x ,po.y);
518
519                 //上矢印__↑（上向き三角と線）を茶色で描画
520                 //茶色CV_RGB (102, 51, 0)
521                 //三角
522                  cvLine(frame, cvPoint(80 + x0, 17 + y0), cvPoint(87 + x0 ,27+ y0),
523                  CV_RGB (102, 51, 0), 12, CV_AA, 0); //A
524                 cvLine(frame, cvPoint(73 + x0, 27+ y0), cvPoint(80+ x0 ,17 + y0),
525                 CV_RGB (102, 51, 0), 12, CV_AA, 0); //B
526                 cvLine(frame, cvPoint(87 + x0,27 + y0), cvPoint(73 + x0 ,27 + y0),
527                  CV_RGB (102, 51, 0), 12, CV_AA, 0); //C
528
529                 count_click = 0;
530         }
531
532
533         //1位置 下↓↓
534         else if(x0<mouth_x && mouth_x<x3 && y4<mouth_y && mouth_y<y5){
535                 po.y += 30;
536                 SetCursorPos(po.x ,po.y);
537
538                 //下矢印__↓（下向き三角と線）を黄色で描画
539                 //黄色CV_RGB (255, 245, 0)
540                 cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
541                  cvPoint(87 + genten_idou_x ,133 + genten_idou_y),
542                  CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
543                 cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
544                  cvPoint(73 + genten_idou_x ,133 + genten_idou_y),
545                  CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
546                 cvLine(frame, cvPoint(87 + genten_idou_x,133 + genten_idou_y),
547                  cvPoint(73 + genten_idou_x ,133 + genten_idou_y),
548                  CV_RGB (255, 245, 0), 12, CV_AA, 0); //
549
550                 count_click = 0;
551         }
552
553         //２位置　下↓↓
554         else if(x0<mouth_x && mouth_x<x1 && y3<mouth_y && mouth_y<y4){
555                 po.y += 30;
556                 SetCursorPos(po.x ,po.y);
557
558                 //下矢印__↓（下向き三角と線）を黄色で描画
559                 //黄色CV_RGB (255, 245, 0)
560                 //三角
561                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
562                  cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
563                  CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
564                 cvLine(frame, cvPoint(190 + genten_idou_x,200 + genten_idou_y),
565                  cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
566                  CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
567                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
568
```

```
569                 cvPoint(190 + genten_idou_x ,200 + genten_idou_y),
570                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //
571                 //線
572                 cvLine(frame, cvPoint(190 + genten_idou_x,180 + genten_idou_y),
573                 cvPoint(210 + genten_idou_x ,180 + genten_idou_y),
574                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
575
576                 count_click = 0;
577         }
578
579         //３位置　下↓↓
580         else if(x2<mouth_x && mouth_x<x3 && y3<mouth_y && mouth_y<y4){
581                 po.y += 30;
582                 SetCursorPos(po.x ,po.y);
583
584                 //下矢印__↓（下向き三角と線）を黄色で描画
585                 //黄色CV_RGB (255, 245, 0)
586                 //三角
587                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
588                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
589                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
590                 cvLine(frame, cvPoint(190 + genten_idou_x,200 + genten_idou_y),
591                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
592                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
593                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
594                 cvPoint(190 + genten_idou_x ,200 + genten_idou_y),
595                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //
596                 //線
597                 cvLine(frame, cvPoint(190 + genten_idou_x,180 + genten_idou_y),
598                 cvPoint(210 + genten_idou_x ,180 + genten_idou_y),
599                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
600
601                 count_click = 0;
602         }
603
604
605         //停止
606         else if(x1<mouth_x && mouth_x<x2 && y2<mouth_y && mouth_y<y4){
607                 count_click = 0;
608         }
609
610         //クリック範囲
611         else if(x1<mouth_x && mouth_x<x2 &&    y1<mouth_y && mouth_y<y2){
612
613                 //10秒間カウント
614                 if (count_click < 10) {
615                         count_click = count_click+1; //count_clickを増やす
616                 }else{ // count_clickが回以上なら以下を処理
617                 //マウスクリック(2回→ダブルクリック)
618                 mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
619                 mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
620
621
```

```
622                 mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
623                 mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
624
625                 /*
626                  中心よりやや上に緑色の●を描画
627                 赤色CV_RGB CV_RGB(255, 40, 0)
628                 緑色CV_RGB CV_RGB(53, 161, 107)
629                 口は下の方にあるため,首の可動域は左右よりも上下の方が小さい
630                  */
631                 cvCircle(frame /* 描画するフレームの指定*/,
632                  cvPoint(tyuushin_x, tyuushin_y-10) /* 円の中心（顔の中心）座標*/,
633                 3/* 円の大きさ*/, CV_RGB(53, 161, 107) /* 色*/,
634                  22 /* 線の太さ*/, CV_AA, 0);
635
636
637                 count_click = 0;
638         }
639
640         //停止範囲　y_c1<y<y2
641         }else{
642         //停止　何もしない
643         }
644         /*
645         //向きA　左←青
646         if(ave_y < 3*ave_x /4 && ave_y >= -3*ave_x /4){
647         }
648         //向きC　右→赤
649         else if(ave_y <= -3*ave_x /4 && ave_y > 3*ave_x /4){
650         }
651         //向きD　上↑緑
652         else if(ave_y <= 3*ave_x /4 && ave_y < -3*ave_x /4){
653         }
654         //向きE　下↓黄色
655         else if(ave_y > 3*ave_x /4 && ave_y >= -3*ave_x /4){
656         }*/
657         //}
658
659
660         if(movingState==STOPPING){
661                 stopping_count++;
662         }
663         if(movingState==MOVING_RIGHT){
664                 right_count++;
665         }
666         if(movingState==MOVING_LEFT){
667                 left_count++;
668         }
669         if(movingState==MOVING_UP){
670                 up_count++;
671         }
672         if(movingState==MOVING_DOWN){
673                 down_count++;
674         }
```

```
675
676
677             //cvFlip(img_out, img_out, 1);                              // 左右反転（鏡面モード）
678             //cvShowImage("オプティカルフロー", img_out);
679             cvCopy(frame_now, frame_pre_face);          // 現在画像を事前画像にコピー
680             }
681
682             cvSetImageROI(frame,cvRect(x0,y0,x3 - genten_idou_x,
683               y5 - genten_idou_y));//ROI指定(切り出し)
684             cvFlip(frame, frame, 1);                              // 左右反転（鏡面モード）
685             cvShowImage ("顔検出", frame);    // 検出結果表示
686             cvResetImageROI(frame);//これをしないとROIのみ表示される
687
688             if(cvWaitKey(33) == 27) break;      // ESCキーを押した時終了
689             }
690
691             // ウィンドウ・キャプチャ・画像の解放
692             //cvDestroyWindow("モーションテンプレート");
693             cvDestroyAllWindows();
694
695             cvReleaseImage(&img_out);
696             cvReleaseImage(&img_tmp1);
697             cvReleaseImage(&img_tmp2);
698             cvReleaseImage(&pyramid_pre);
699             cvReleaseImage(&pyramid_now);
700
701             cvReleaseCapture(&src);
702 }
703
704
705 /* コールバック関数*/
706 void on_mouse (int event, int x, int y, int flags, void *param = NULL){
707             char str[64];    static int line = 0;    const int max_line = 15, w = 15, h = 30;
708
709             // (4)マウスイベントを取得
710             switch (event)
711             {
712             //MOVEと書いてあるけど、マウスが何もしていないとき
713             case CV_EVENT_MOUSEMOVE:
714                         _snprintf(str, 64, "(%d,%d) %s", x, y, "MOUSE_MOVE");
715             break;
716
717             //左ボタンが押されている
718             case CV_EVENT_LBUTTONDOWN:
719                         _snprintf(str, 64, "(%d,%d) %s", x, y, "LBUTTON_DOWN");
720             break;
721
722             //右ボタンが押されている
723             case CV_EVENT_RBUTTONDOWN:
724                         _snprintf(str, 64, "(%d,%d) %s", x, y, "RBUTTON_DOWN");
725             break;
726
727
```

```
728             //中ボタンが押されている
729             case CV_EVENT_MBUTTONDOWN:
730                     _snprintf(str, 64, "(%d,%d) %s", x, y, "MBUTTON_DOWN");
731             break;
732
733             //左ボタンが離される
734             case CV_EVENT_LBUTTONUP:
735                     _snprintf(str, 64, "(%d,%d) %s", x, y, "LBUTTON_UP");
736             break;
737
738             //右ボタンが離される
739             case CV_EVENT_RBUTTONUP:
740                     _snprintf(str, 64, "(%d,%d) %s", x, y, "RBUTTON_UP");
741             break;
742
743             //中ボタンが離される
744             case CV_EVENT_MBUTTONUP:
745                     _snprintf(str, 64, "(%d,%d) %s", x, y, "MBUTTON_UP");
746             break;
747
748             //左ボダンがダブルクリックされる
749             case CV_EVENT_LBUTTONDBLCLK:
750                     _snprintf(str, 64, "(%d,%d) %s", x, y, "LBUTTON_DOUBLE_CLICK");
751             break;
752
753             // 右ボダンがダブルクリックされる
754             case CV_EVENT_RBUTTONDBLCLK:
755                     _snprintf(str, 64, "(%d,%d) %s", x, y, "RBUTTON_DOUBLE_CLICK");
756             break;
757
758             //中ボダンがダブルクリックされる
759             case CV_EVENT_MBUTTONDBLCLK:
760                     _snprintf(str, 64, "(%d,%d) %s", x, y,
761 "MBUTTON_DOUBLE_CLICK");
762             break;
763             }
764
765             // マウスボタン，修飾キーを取得
766             if (flags & CV_EVENT_FLAG_LBUTTON)strncat (str, " + LBUTTON", 64);
767             if (flags & CV_EVENT_FLAG_RBUTTON)strncat (str, " + RBUTTON", 64);
768             if (flags & CV_EVENT_FLAG_MBUTTON)strncat (str, " + MBUTTON", 64);
769             if (flags & CV_EVENT_FLAG_CTRLKEY) strncat (str, " + CTRL", 64);
770             if (flags & CV_EVENT_FLAG_SHIFTKEY) strncat (str, " + SHIFT", 64);
771             if (flags & CV_EVENT_FLAG_ALTKEY)strncat (str, " + ALT", 64);
772
773             if(frame != 0){
774                 // マウス座標，イベント，修飾キーなどを画像に描画，表示
775                 if (line > max_line) {
776                         cvGetRectSubPix (img, img, cvPoint2D32f (320 - 0.5, 240 - 0.5 + h));
777                         cvPutText (img, str, cvPoint (w, 20 + h * max_line),
778                           &font, CV_RGB (0, 200, 100));
779                 } else {
780                         cvPutText (img, str, cvPoint (w, 20 + h * line),
```

```
781                         &font, CV_RGB (0, 200, 100));
782                 }
783                 line++;
784                 cvShowImage ("MOUSE", img);
785             }
786 }
787
788
789
```

iii－2 Windows タブレットの顔の位置によるマウス操作のプログラム

```
1   // オプティカルフロー.cpp : コンソールアプリケーションのエントリポイントを定義し
2   ます。
3   //
4
5   #include "stdafx.h"
6   #include <cv.h>
7   #include <highgui.h>
8   #include <windows.h>
9
10  #pragma comment(lib,"cv.lib")
11  #pragma comment(lib,"cxcore.lib")
12  #pragma comment(lib,"highgui.lib")
13
14  #include <time.h>           // 時間関連ヘッダ
15  #include <math.h>           // 数学関連ヘッダ
16
17  /* グローバル変数*/
18  CvFont font;
19  IplImage *frame=0;          // 画像変数宣言
20  IplImage *img = 0;
21  /* プロトタイプ宣言*/
22  void on_mouse (int event, int x, int y, int flags, void *param);
23  int _tmain(int argc, _TCHAR* argv[])
24  {
25          CvSeq *objects;
26          CvSeq *objects2;
27
28          CvRect *r;
29          CvRect lower_half;
30
31          char *cascade_name;     // カスケードファイル名
32          CvCapture* src;         // ビデオキャプチャ宣言
33
34          IplImage *frame_copy=0;         //フレーム単位データコピー用
35          IplImage *gray=0;
36          IplImage *detect_frame=0;
37
38          // 画像変数宣言
39          IplImage *frame_pre=0;
40          IplImage *frame_now=0;
41
42          IplImage *frame_pre_face=0;
43          IplImage *img_out=0;
44          IplImage *img_tmp1=0;
45          IplImage *img_tmp2=0;
46          IplImage *pyramid_now=0;
47          IplImage *pyramid_pre=0;
48
```

```
49
50          // 反復アルゴリズム用終了条件
51          CvTermCriteria criteria;
52          criteria = cvTermCriteria(CV_TERMCRIT_ITER
53          | CV_TERMCRIT_EPS, 20, 0.03);
54
55          CvHaarClassifierCascade *cascade;          // カスケード分類器の宣言
56          CvMemStorage *storage;                     // メモリ領域の宣言
57
58          // カスケード・ファイルの読み込み
59          cascade_name = "haarcascade_frontalface_alt_tree.xml";
60          cascade = (CvHaarClassifierCascade *) cvLoad(cascade_name, 0, 0, 0);
61          storage = cvCreateMemStorage (0);          // メモリ領域の確保
62          cvClearMemStorage(storage);                // メモリ領域の初期化
63
64
65          /*
66          カメラの選択
67          Windowsタブレッド
68          0=外カメラ(rear camera)、1=内カメラ(front camera)、2=USBカメラ（外付け）
69          PC（PCにカメラがついてない場合）
70          0=USBカメラ（外付け）
71          */
72          src = cvCaptureFromCAM(0);                 // 映像取得（カメラ映像）
73
74          //カメラから取得した画像のサイズ指定
75          cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_WIDTH, 640);
76          cvSetCaptureProperty(src, CV_CAP_PROP_FRAME_HEIGHT, 480);
77
78          if(src == NULL){printf("映像が取得できません。¥n"); cvWaitKey(0);
79          return -1;}
80          cvNamedWindow ("顔検出");
81
82          int STOPPING = 0;
83
84          //繰り返しカウント(時間)
85          int count_click = 0;
86
87          while(1){
88            // 1フレーム取得
89            frame = cvQueryFrame(src); if(frame == NULL) break;
90            if(frame_pre==0) frame_pre = cvCreateImage
91            (cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1);
92            if(frame_now==0) frame_now = cvCreateImage
93            (cvSize(frame->width, frame->height), IPL_DEPTH_8U, 1);
94
95            if(img_out==0) img_out = cvCreateImage
96            (cvSize(frame->width, frame->height), IPL_DEPTH_8U, 3);
97            if(img_tmp1==0) img_tmp= cvCreateImage
98            (cvSize(frame->width, frame->height), IPL_DEPTH_32F, 1);
99            if(img_tmp2==0) img_tmp2= cvCreateImage
100           (cvSize(frame->width, frame->height), IPL_DEPTH_32F, 1);
101
```

```
    if(pyramid_pre==0) pyramid_pre = cvCreateImage
    (cvSize(frame->width+8, frame->height/3),IPL_DEPTH_8U, 1);
    if(pyramid_now==0) pyramid_now = cvCreateImage
    (cvSize(frame->width+8, frame->height/3),IPL_DEPTH_8U, 1);

    if(frame_copy==0) frame_copy = cvCreateImage(cvSize(frame->width,
    frame->height), IPL_DEPTH_8U, frame->nChannels);
    if(frame->origin == IPL_ORIGIN_TL) {
        cvCopy(frame, frame_copy);
    } else {
        cvFlip(frame, frame_copy);
    }

cvCvtColor(frame, frame_now, CV_BGR2GRAY); // グレースケール変換

if(frame_pre_face==0){
        frame_pre_face = cvCreateImage(cvSize(frame->width,
        frame->height), IPL_DEPTH_8U, 1);
        cvCopy(frame_now, frame_pre_face);
 //はじめの１回はframe_pre_faceが無いのでframe_nowを
 //frame_pre_faceにする為にコピーする
}

// 読み込んだ画像のグレースケール化、及びヒストグラムの均一化を行う
if(gray==0) gray = cvCreateImage
(cvSize(frame_copy->width, frame_copy->height), IPL_DEPTH_8U, 1);
if(detect_frame==0) detect_frame = cvCreateImage
(cvSize((frame_copy->width), (frame_copy->height)), IPL_DEPTH_8U, 1);
cvResetImageROI(detect_frame);
cvCvtColor(frame_copy, gray, CV_BGR2GRAY);
cvResize(gray, detect_frame, CV_INTER_LINEAR);
cvEqualizeHist(detect_frame, detect_frame);

//移動範囲の視覚化
//原点を移動する
//上に移動の際、顔が切れて認識しなくなるため
int genten_idou_x = 240;
int genten_idou_y = 160;

//切り替え範囲x、y
int x0 = 0    + genten_idou_x;
int x1 = 57 + genten_idou_x;
int x2 = 103 + genten_idou_x;
int x3 = 160 + genten_idou_x;

int y0 = 0    + genten_idou_y;
int y1 = 45   + genten_idou_y;
int y2 = 85 + genten_idou_y;
int y3 = 115 + genten_idou_y;
int y4 = 120 + genten_idou_y;
int y5 = 160 + genten_idou_y;

int l1x = (x2 + 25);
```

```
155             int r1x = (x1-25) ;
156
157             //矢印の変数
158             int saunkaku_x = 10;
159             int saunkaku_y = 14 /2;
160             int saunkaku_h = 5;
161
162             //顔検出のフレームの中心座標
163             int tyuushin_x = 80+ genten_idou_x;
164             int tyuushin_y = 65+ genten_idou_y;
165
166             //範囲を灰色で表示（x,y）
167             cvLine(frame, cvPoint(x1,y1), cvPoint(x1,y4), CV_RGB (200, 200, 203), 4,
168             CV_AA, 0); //T1
169             cvLine(frame, cvPoint(x2,y1), cvPoint(x2,y4), CV_RGB (200, 200, 203), 4,
170             CV_AA, 0); //T2
171
172             cvLine(frame, cvPoint(x0,y1), cvPoint(x3,y1), CV_RGB (200, 200, 203), 4,
173             CV_AA, 0); //Y1
174             cvLine(frame, cvPoint(x1,y2), cvPoint(x2,y2), CV_RGB (200, 200, 203), 4,
175             CV_AA, 0); //Y1-2
176             cvLine(frame, cvPoint(x0,y3), cvPoint(x1,y3), CV_RGB (200, 200, 203), 4,
177             CV_AA, 0); //Y2a
178             cvLine(frame, cvPoint(x2,y3), cvPoint(x3,y3), CV_RGB (200, 200, 203), 4,
179             CV_AA, 0); //Y2b
180             cvLine(frame, cvPoint(x1,y4), cvPoint(x2,y4), CV_RGB (200, 200, 203), 4,
181             CV_AA, 0); //Y2-3
182
183 //cvLine(frame, cvPoint(r1x,y1), cvPoint(r1x,y3), CV_RGB (255, 0, 0), 4, CV_AA, 0);
184 //クリックとの境目　y_c1
185 //cvLine(frame, cvPoint(x0,y1), cvPoint(x0,y3), CV_RGB (0, 255, 0), 4, CV_AA, 0);
186 //クリックとの境目　y_c1
187 //cvLine(frame, cvPoint(x1,y1), cvPoint(x1,y3), CV_RGB (0, 0, 255), 4, CV_AA, 0);
188 //クリックとの境目　y_c1
189
190             //ただし、反転注意（右三角を描くと左三角を表示）
191             //停止
192             //中心よりやや上に灰色の○を描画
193             //口は下の方にあるため首の可動域は左右よりも上下の方が小さい
194             cvCircle(frame /* 描画するフレームの指定*/,
195             cvPoint(tyuushin_x, tyuushin_y) /* 円の中心（顔の中心）座標*/,
196             14/* 円の大きさ*/, CV_RGB(200, 200, 203) /* 色*/, 2 /* 線の太さ*/,
197             CV_AA, 0);
198
199             //右矢印｜→（右向き三角と線）を灰色で描画
200             //ピンク色CV_RGB (255, 153, 160)
201             //三角
202             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
203             cvPoint(32 + genten_idou_x ,80 + genten_idou_y), CV_RGB (200, 200, 203),
204             12, CV_AA, 0); //
205             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
206             cvPoint(32 + genten_idou_x ,90+ genten_idou_y), CV_RGB (200, 200, 203),
207
```

```
208             12, CV_AA, 0); // |
209             cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
210             cvPoint(32+ genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
211             12, CV_AA, 0); //／
212
213             //左矢印← |（左向き三角と線）を灰色で描画
214             ////空色CV_RGB (102, 204, 255)
215             cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
216             cvPoint(128 + genten_idou_x ,80 + genten_idou_y), CV_RGB (200, 200, 203),
217             12, CV_AA, 0); //
218             cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
219             cvPoint(128 + genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
220             12, CV_AA, 0); //__
221             cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
222             cvPoint(128 + genten_idou_x ,90 + genten_idou_y), CV_RGB (200, 200, 203),
223             12, CV_AA, 0); //／
224
225             //上矢印__↑（上向き三角と線）を灰色で描画
226             //茶色CV_RGB (102, 51, 0)
227             cvLine(frame, cvPoint(80 + x0, 17 + y0), cvPoint(87 + x0 ,27+ y0),
228             CV_RGB (200, 200, 203), 12, CV_AA, 0); //A
229             cvLine(frame, cvPoint(73 + x0, 27+ y0), cvPoint(80+ x0 ,17 + y0),
230             CV_RGB (200, 200, 203), 12, CV_AA, 0); //B
231             cvLine(frame, cvPoint(87 + x0,27 + y0), cvPoint(73 + x0 ,27 + y0),
232             CV_RGB (200, 200, 203), 12, CV_AA, 0); //C
233
234             //下矢印__↓（下向き三角と線）を灰色で描画
235             //黄色CV_RGB (255, 245, 0)
236             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
237             cvPoint(87 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
238             12, CV_AA, 0); //／
239             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
240             cvPoint(73 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
241             12, CV_AA, 0); //__
242             cvLine(frame, cvPoint(87 + genten_idou_x,133 + genten_idou_y),
243             cvPoint(73 + genten_idou_x ,133 + genten_idou_y), CV_RGB (200, 200, 203),
244             12, CV_AA, 0); //
245
246
247             // Haar-Like顔検出
248             objects = cvHaarDetectObjects
249             (detect_frame, cascade, storage, 1.1, 3, 0, cvSize(0, 0));
250
251             int face_max_size = 0;
252             int face_max_index = -1;
253             CvRect *r_face_max=0;
254
255             for (int i = 0; i < (objects ? objects->total : 0); i++) {   // 検出数の繰り返し
256               // 検出矩形領域の取得rは顔の描画の起点座標
257               r = (CvRect *) cvGetSeqElem(objects, i);
258
259               //一番大きな顔を使用
260
```

```
261             int length = r->width + r->height;
262
263             if(length > face_max_size){
264                     face_max_size = length;
265                     r_face_max = r;
266                     face_max_index = i;
267              }
268         }
269
270         if(r_face_max!=0){
271          //顔の(x,y)=(1/2, 3/4)に赤色の●を描画
272          cvCircle(frame /* 描画するフレームの指定*/,
273          cvPoint(r_face_max->x + r_face_max->width/2,
274          r_face_max->y + 3 *(r_face_max->height)/4) /* 円の中心（顔の中心）座標*/,
275          2/* 円の大きさ*/, CV_RGB(255, 40, 0) /* 色*/, 16 /* 線の太さ*/, CV_AA, 0);
276
277          // この時点で、最大の顔のオブジェクトの矩形領域はr_face_max
278          //顔の下半分の領域
279          lower_half = cvRect(r_face_max->x + r_face_max->width / 3,
280          r_face_max->y + 2*r_face_max->height / 3,
281          //原点から/3幅x,2/3の高さyをずらす①
282          r_face_max->width/3, r_face_max->height / 3); //縮尺
283
284          cvResetImageROI(detect_frame);
285          cvSetImageROI(detect_frame, lower_half);
286
287          POINT po;//カーソル位置用
288          GetCursorPos(&po);//マウスの座標を取得
289
290         // 顔の●の定義
291         int mouth_x = (r_face_max->x + r_face_max->width/2);
292         int mouth_y = (r_face_max->y + 3 *(r_face_max->height)/4);
293
294
295         //左←←
296         if(x2<mouth_x && mouth_x<l1x && y1<mouth_y && mouth_y<y3){
297            po.x -= 30;
298            SetCursorPos(po.x ,po.y);
299
300            //左矢印←｜（左向き三角と線）を空色で描画
301            //空色CV_RGB (102, 204, 255)
302            //三角
303            cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
304           cvPoint(128 + genten_idou_x ,80 + genten_idou_y),
305           CV_RGB (102, 204, 255), 12, CV_AA, 0); //
306            cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
307            cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
308            CV_RGB (102, 204, 255), 12, CV_AA, 0); //__
309            cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
310            cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
311            CV_RGB (102, 204, 255), 12, CV_AA, 0); //／
312
313
```

```
314             count_click = 0;
315         }
316
317         //左←
318         else if(l1x<mouth_x && mouth_x<x3 && y1<mouth_y && mouth_y<y3){
319             po.x -= 15;
320             SetCursorPos(po.x ,po.y);
321
322             //左矢印←（左向き三角）を空色で描画
323             //空色CV_RGB (102, 204, 255)
324             cvLine(frame, cvPoint(142    + genten_idou_x, 85 + genten_idou_y),
325             cvPoint(128 + genten_idou_x ,80 + genten_idou_y),
326             CV_RGB (102, 204, 255), 12, CV_AA, 0); //
327             cvLine(frame, cvPoint(142 + genten_idou_x,85 + genten_idou_y),
328             cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
329             CV_RGB (102, 204, 255), 12, CV_AA, 0); //__
330             cvLine(frame, cvPoint(128 + genten_idou_x,80 + genten_idou_y),
331             cvPoint(128 + genten_idou_x ,90 + genten_idou_y),
332             CV_RGB (102, 204, 255), 12, CV_AA, 0); //／
333
334             count_click = 0;
335         }
336
337
338         //右→→
339         else if(x0<mouth_x && mouth_x<r1x && y1<mouth_y && mouth_y<y3){
340             po.x += 30;
341             SetCursorPos(po.x ,po.y);
342
343             //右矢印｜→（右向き三角と線）をピンク色で描画
344             //ピンク色CV_RGB (255, 153, 160)
345             //三角
346             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
347             cvPoint(32 + genten_idou_x ,80 + genten_idou_y), CV_RGB(255, 153,
348             160) , 12, CV_AA, 0); //
349             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
350             cvPoint(32 + genten_idou_x ,90+ genten_idou_y), CV_RGB (255, 153,
351             160) , 12, CV_AA, 0); // |
352             cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
353             cvPoint(32+ genten_idou_x ,90 + genten_idou_y), CV_RGB (255, 153,
354             160) ,12, CV_AA, 0); //／
355
356             count_click = 0;
357         }
358
359
360         //右→
361         else if (r1x<mouth_x && mouth_x<x1 && y1<mouth_y && mouth_y<y3){
362             po.x += 15;
363             SetCursorPos(po.x ,po.y);
364
365             //右矢印→（右向き三角）をピンク色で描画
366
```

```
367             //ピンク色CV_RGB (255, 153, 160)
368             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
369             cvPoint(32 + genten_idou_x ,80 + genten_idou_y),
370             CV_RGB(255, 153, 160) , 12, CV_AA, 0); //
371             cvLine(frame, cvPoint(18 + genten_idou_x,85 + genten_idou_y),
372             cvPoint(32 + genten_idou_x ,90+ genten_idou_y),
373             CV_RGB (255, 153, 160) , 12, CV_AA, 0); // |
374             cvLine(frame, cvPoint(32 + genten_idou_x,80 + genten_idou_y),
375             cvPoint(32+ genten_idou_x ,90 + genten_idou_y),
376             CV_RGB (255, 153, 160) , 12, CV_AA, 0); //／
377
378             count_click = 0;
379         }
380
381
382         //上↑↑
383         else if(x0<mouth_x && mouth_x<x3 && y0<mouth_y && mouth_y<y1){
384             po.y -= 30;
385             SetCursorPos(po.x ,po.y);
386
387             //上矢印__↑（上向き三角と線）を茶色で描画
388             //茶色CV_RGB (102, 51, 0)
389             //三角
390             cvLine(frame, cvPoint(80 + x0, 17 + y0),
391             cvPoint(87 + x0 ,27+ y0), CV_RGB (102, 51, 0), 12, CV_AA, 0); //A
392             cvLine(frame, cvPoint(73 + x0, 27+ y0), cvPoint(80+ x0 ,17 + y0),
393             CV_RGB (102, 51, 0), 12, CV_AA, 0); //B
394             cvLine(frame, cvPoint(87 + x0,27 + y0),
395             cvPoint(73 + x0 ,27 + y0), CV_RGB (102, 51, 0), 12, CV_AA, 0); //C
396
397             count_click = 0;
398         }
399
400
401         //1位置　下↓↓
402         else if(x0<mouth_x && mouth_x<x3 && y4<mouth_y && mouth_y<y5){
403             po.y += 30;
404             SetCursorPos(po.x ,po.y);
405
406             //下矢印__↓（下向き三角と線）を黄色で描画
407             //黄色CV_RGB (255, 245, 0)
408             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
409             cvPoint(87 + genten_idou_x ,133 + genten_idou_y),
410             CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
411             cvLine(frame, cvPoint(80 + genten_idou_x,143 + genten_idou_y),
412             cvPoint(73 + genten_idou_x ,133 + genten_idou_y),
413             CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
414             cvLine(frame, cvPoint(87 + genten_idou_x,133 + genten_idou_y),
415             cvPoint(73 + genten_idou_x ,133 + genten_idou_y),
416             CV_RGB (255, 245, 0), 12, CV_AA, 0); //
417
418             count_click = 0;
419
```

```
420             }
421
422             //２位置　下↓↓
423             else if(x0<mouth_x && mouth_x<x1 && y3<mouth_y && mouth_y<y4){
424                 po.y += 30;
425                 SetCursorPos(po.x ,po.y);
426
427                 //下矢印__↓（下向き三角と線）を黄色で描画
428                 //黄色CV_RGB (255, 245, 0)
429                 //三角
430                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
431                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
432                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
433                 cvLine(frame, cvPoint(190 + genten_idou_x,200 + genten_idou_y),
434                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
435                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
436                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
437                 cvPoint(190 + genten_idou_x ,200 + genten_idou_y),
438                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //
439                 //線
440                 cvLine(frame, cvPoint(190 + genten_idou_x,180 + genten_idou_y),
441                 cvPoint(210 + genten_idou_x ,180 + genten_idou_y),
442                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
443
444                 count_click = 0;
445             }
446
447             //３位置　下↓↓
448             else if(x2<mouth_x && mouth_x<x3 && y3<mouth_y && mouth_y<y4){
449                 po.y += 30;
450                 SetCursorPos(po.x ,po.y);
451
452                 //下矢印__↓（下向き三角と線）を黄色で描画
453                 //黄色CV_RGB (255, 245, 0)
454                 //三角
455                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
456                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
457                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //／
458                 cvLine(frame, cvPoint(190 + genten_idou_x,200 + genten_idou_y),
459                 cvPoint(210 + genten_idou_x ,200 + genten_idou_y),
460                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
461                 cvLine(frame, cvPoint(200 + genten_idou_x,220 + genten_idou_y),
462                 cvPoint(190 + genten_idou_x ,200 + genten_idou_y),
463                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //
464                   //線
465                 cvLine(frame, cvPoint(190 + genten_idou_x,180 + genten_idou_y),
466                 cvPoint(210 + genten_idou_x ,180 + genten_idou_y),
467                 CV_RGB (255, 245, 0), 12, CV_AA, 0); //__
468
469                 count_click = 0;
470             }
471
472
```

```
473
474         //停止
475         else if(x1<mouth_x && mouth_x<x2 && y2<mouth_y && mouth_y<y4){
476             count_click = 0;
477         }
478
479
480         //クリック範囲
481         else if(x1<mouth_x && mouth_x<x2 &&    y1<mouth_y && mouth_y<y2){
482             //10秒間カウント
483             if (count_click < 10) {
484                     count_click = count_click+1; //count_clickを増やす
485             }else{ // count_clickが回以上なら以下を処理
486             //マウスクリック(2回→ダブルクリック)
487             mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
488             mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
489
490             mouse_event( MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
491             mouse_event( MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
492
493         /*
494         中心よりやや上に緑色の●を描画
495         赤色CV_RGB CV_RGB(255, 40, 0)
496         緑色CV_RGB CV_RGB(53, 161, 107)
497         口は下の方にあるため,首の可動域は左右よりも上下の方が小さい
498         */
499             cvCircle(frame /* 描画するフレームの指定*/,
500             cvPoint(tyuushin_x, tyuushin_y-10) /* 円の中心（顔の中心）座標*/,
501             3/* 円の大きさ*/, CV_RGB(53, 161, 107) /* 色*/, 22 /* 線の太さ*/,
502             CV_AA, 0);
503
504             count_click = 0;
505             }
506         }
507
508         cvCopy(frame_now, frame_pre_face);          // 現在画像を事前画像にコピー
509         }
510
511         cvSetImageROI(frame,cvRect(x0,y0,x3 - genten_idou_x,y5
512         - genten_idou_y));//ROI指定(切り出し)
513         cvFlip(frame, frame, 1);                    // 左右反転（鏡面モード）
514         cvShowImage ("顔検出", frame);    // 検出結果表示
515         cvResetImageROI(frame);//これをしないとROIのみ表示される
516
517         if(cvWaitKey(33) == 27) break;      // ESCキーを押した時終了
518         }
519
520         cvDestroyAllWindows();
521
522         cvReleaseImage(&img_out);
523         cvReleaseImage(&img_tmp1);
524         cvReleaseImage(&img_tmp2);
525
```

```
526             cvReleaseImage(&pyramid_pre);
527             cvReleaseImage(&pyramid_now);
528
529             cvReleaseCapture(&src);
530     }
531
532
533
```

Ⅳ 回路図と配線図

4.2.1 節、6.2.1 節にて作成した回路の回路図と配線図を以下に示す。

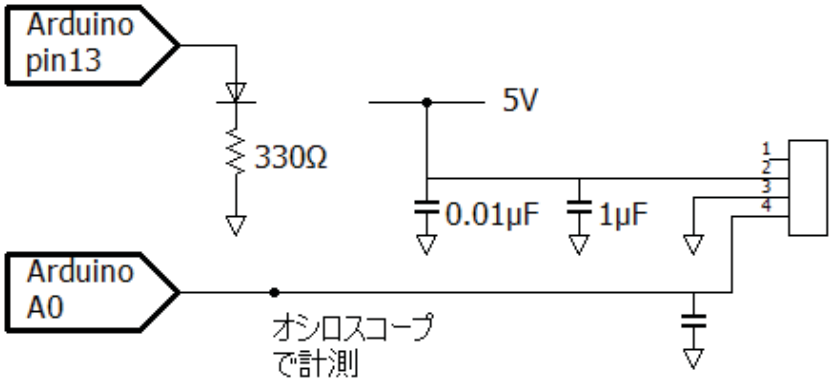


図 iv.1: 回路図

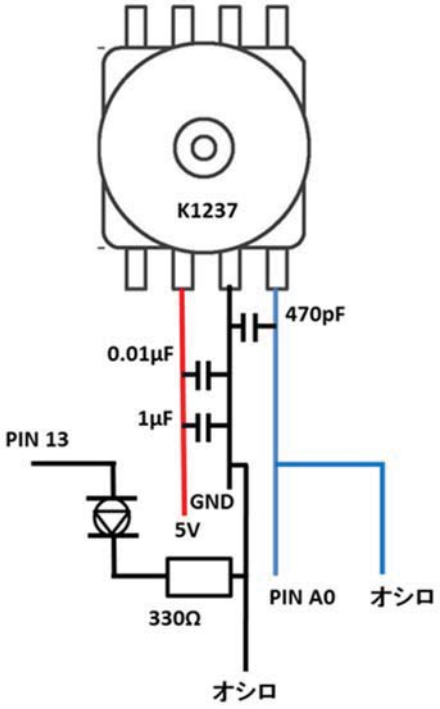
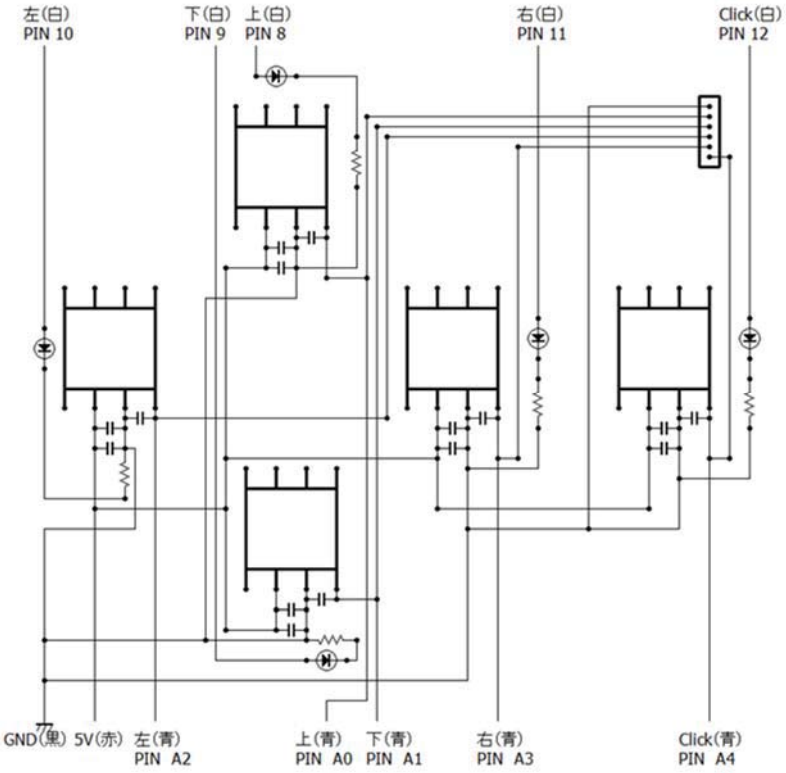


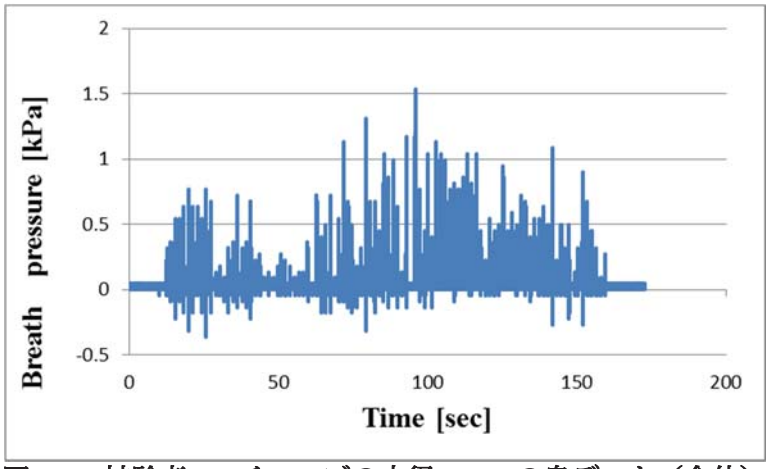
図 iv.2: 配線図 1


左(白)
PIN 10
下(白)
PIN 9
上(白)
PIN 8
右(白)
PIN 11
Click(白)
PIN 12
GND(黒)
5V(赤)
左(青)
PIN A2
上(青)
PIN A0
下(青)
PIN A1
右(青)
PIN A3
Click(青)
PIN A4

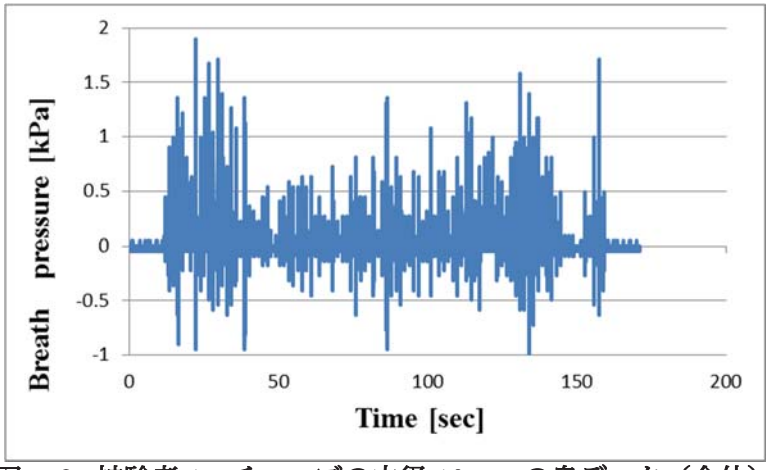
図iv.3: 配線図 2

Ⅴ 息のデータ

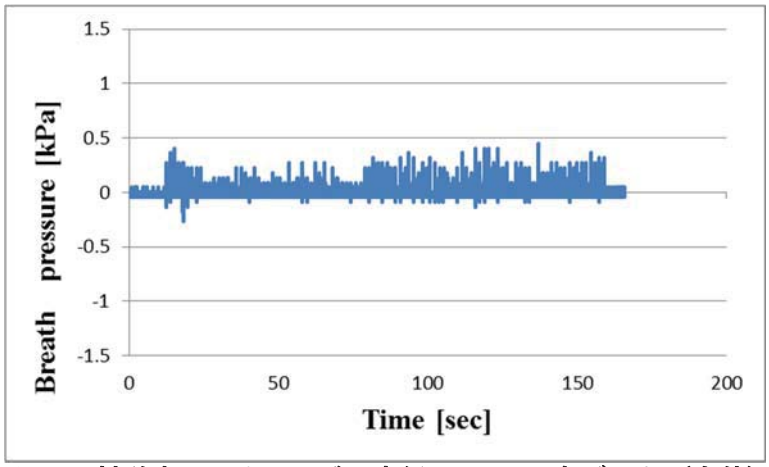
4.2.3 節の実験の被験者 5 名の息の電圧を圧力に変換したものを図ⅴ.1～図ⅴ.30 に記す。



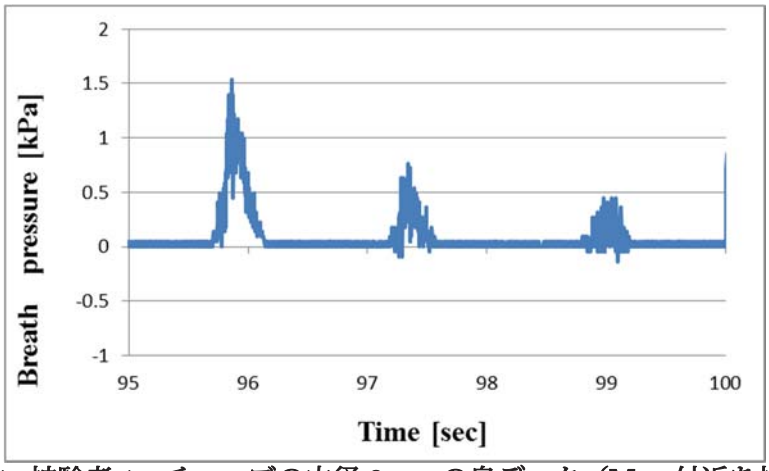
図ⅴ.1: 被験者 1　チューブの内径 2mm の息データ（全体）



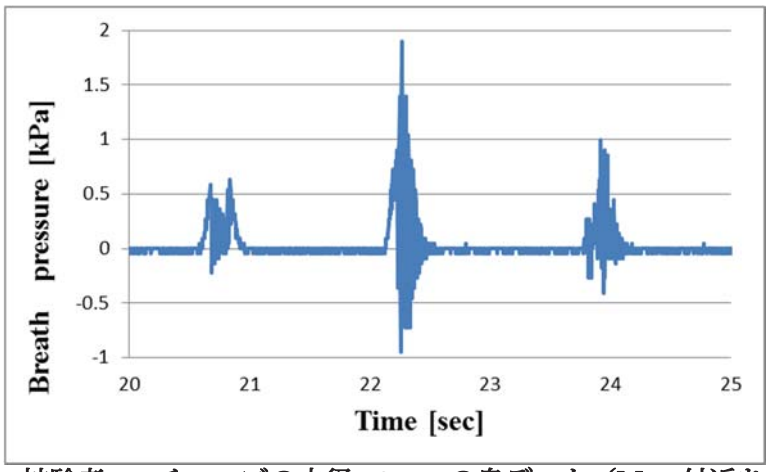
図ⅴ.2: 被験者 1　チューブの内径 10mm の息データ（全体）



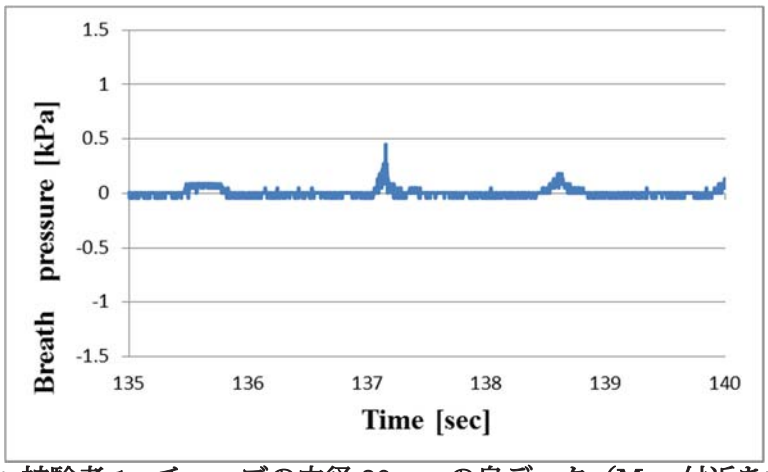
図ⅴ.3: 被験者 1　チューブの内径 30mm の息データ（全体）



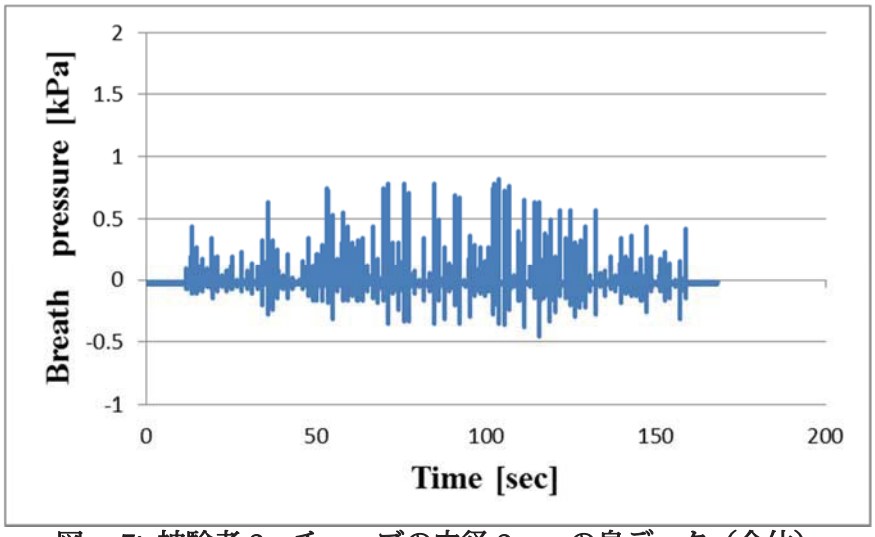
図ｖ.4: 被験者１　チューブの内径 2mm の息データ（Max 付近を拡大）



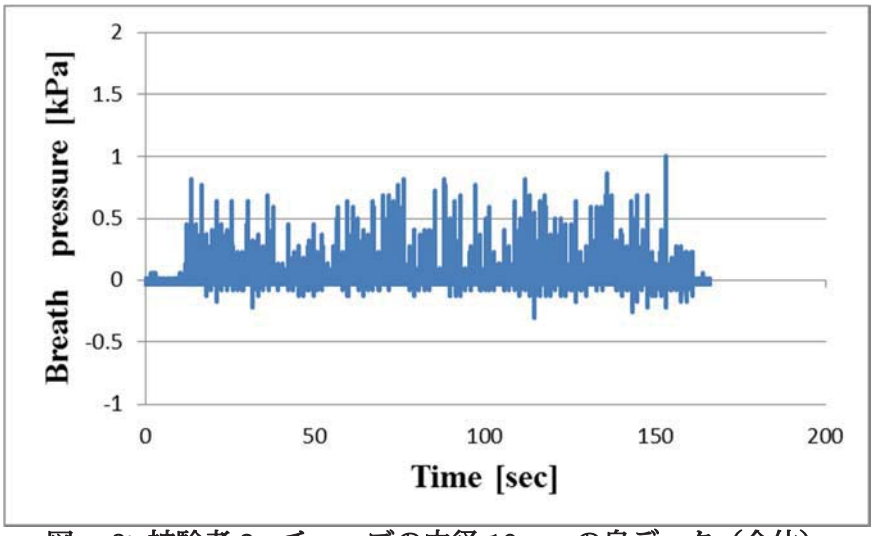
図ｖ.5: 被験者１　チューブの内径 10mm の息データ（Max 付近を拡大）



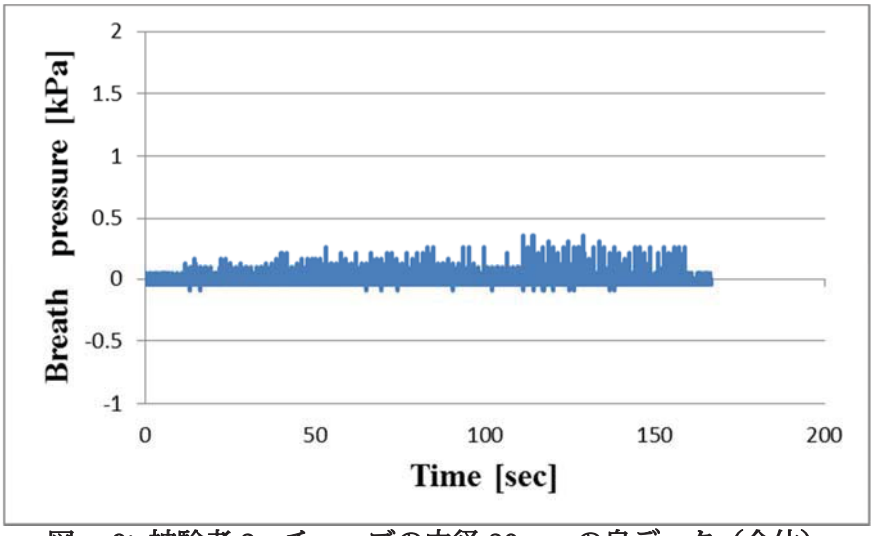
図ｖ.6: 被験者１　チューブの内径 30mm の息データ（Max 付近を拡大）



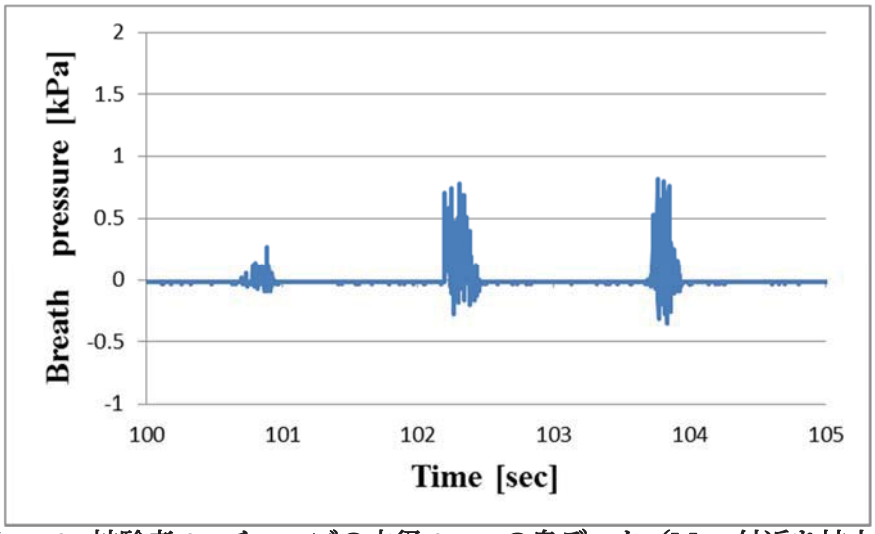
図ｖ.7: 被験者 2　チューブの内径 2mm の息データ（全体）



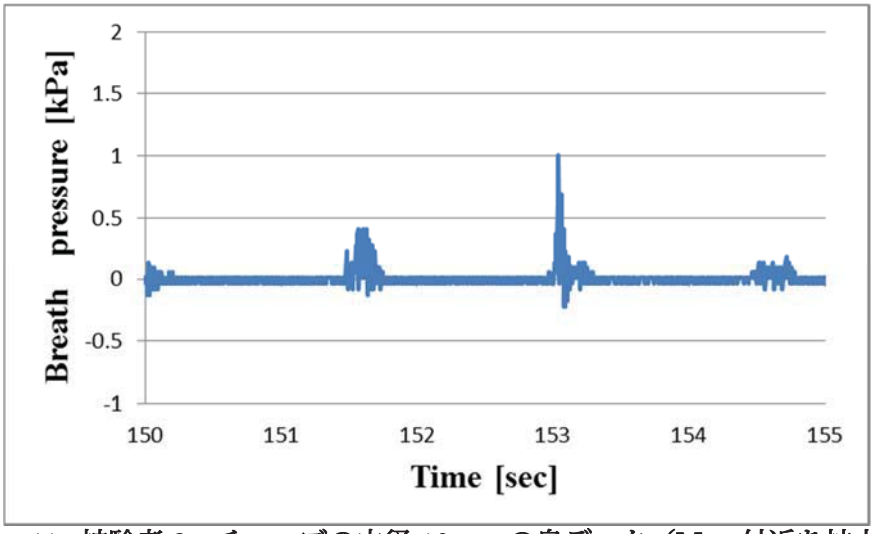
図ｖ.8: 被験者 2　チューブの内径 10mm の息データ（全体）



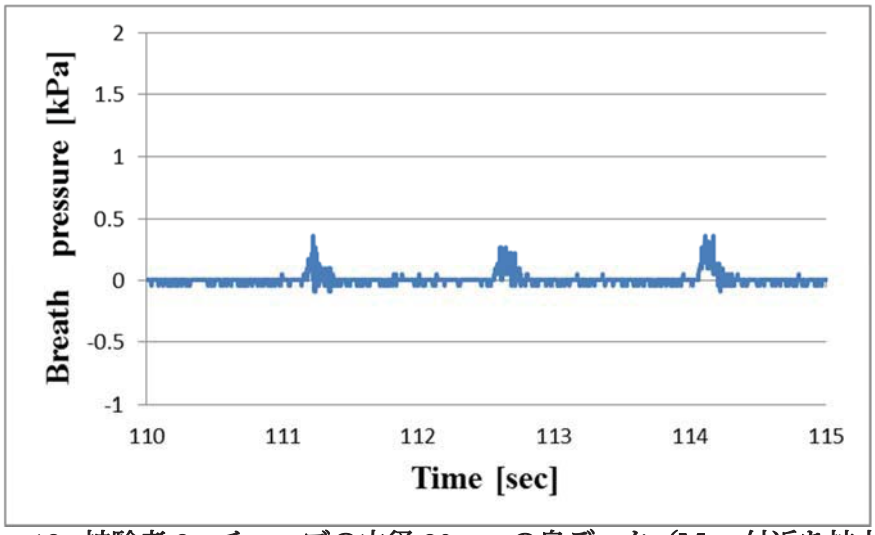
図ｖ.9: 被験者 2　チューブの内径 30mm の息データ（全体）



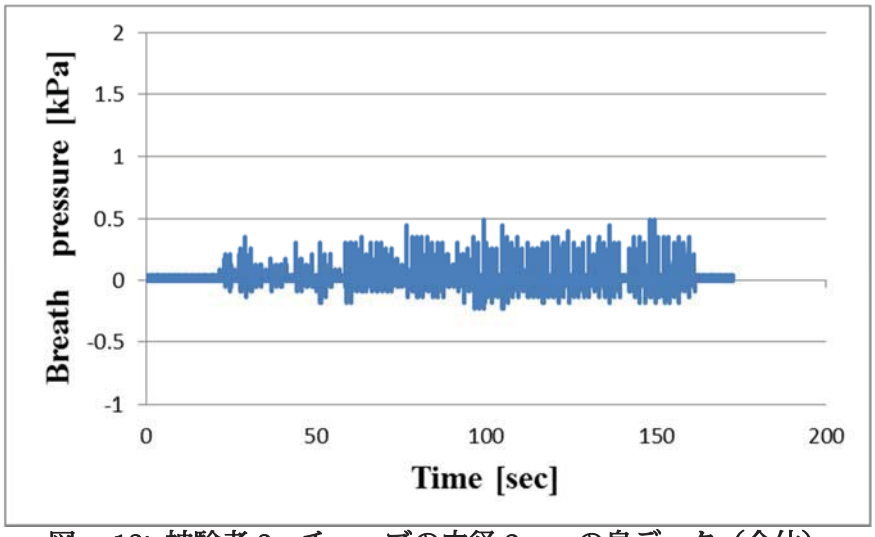
図ｖ.10: 被験者 2　チューブの内径 2mm の息データ（Max 付近を拡大）



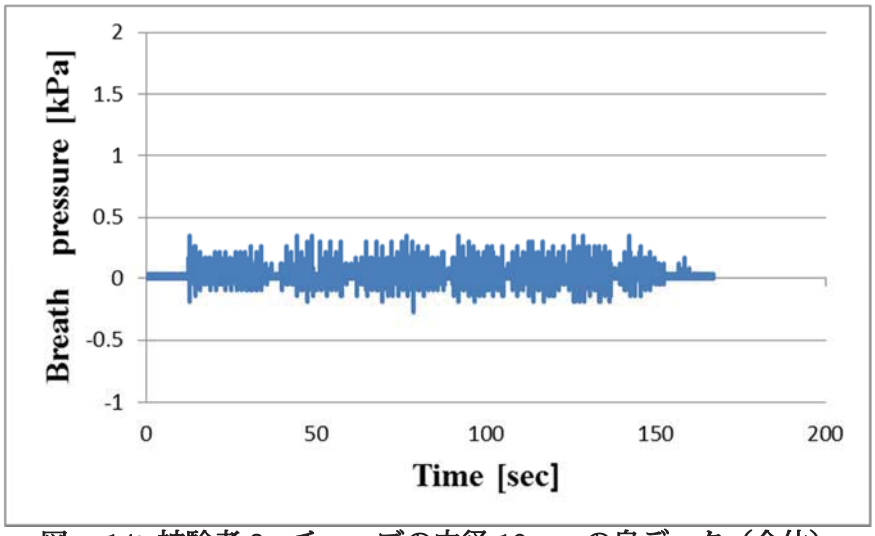
図ｖ.11: 被験者 2　チューブの内径 10mm の息データ（Max 付近を拡大）



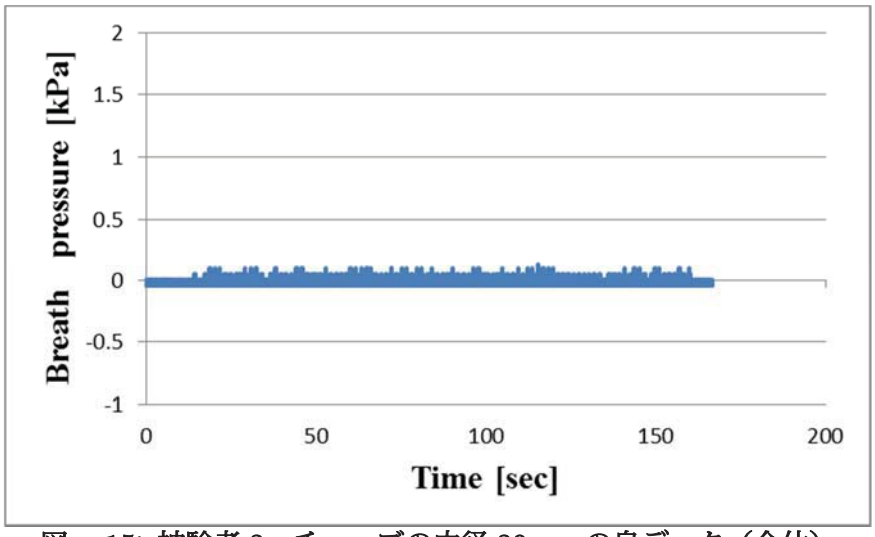
図ｖ.12: 被験者 2　チューブの内径 30mm の息データ（Max 付近を拡大）



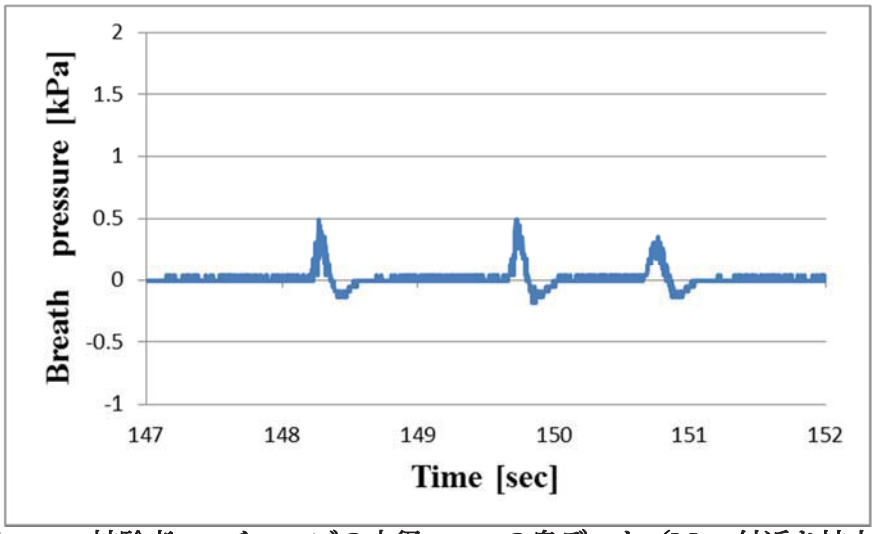
図ｖ.13: 被験者 3　チューブの内径 2mm の息データ（全体）



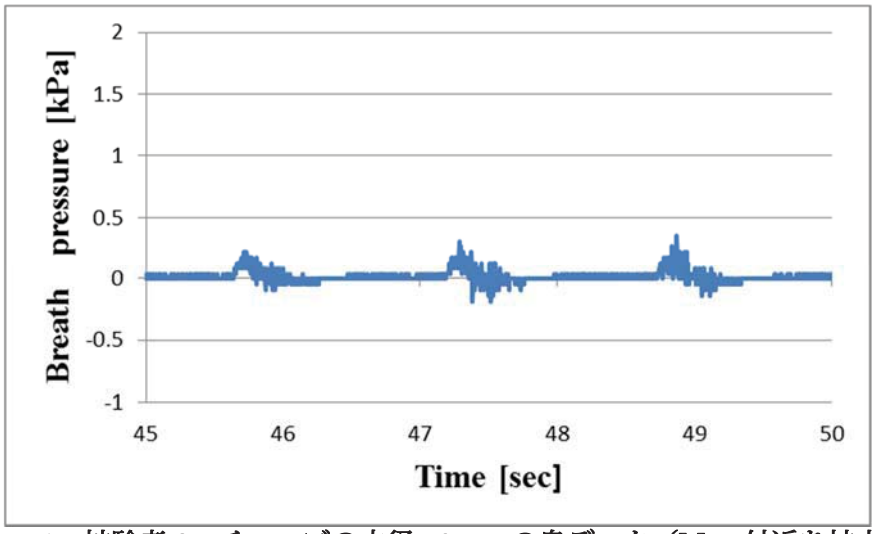
図ｖ.14: 被験者 3　チューブの内径 10mm の息データ（全体）



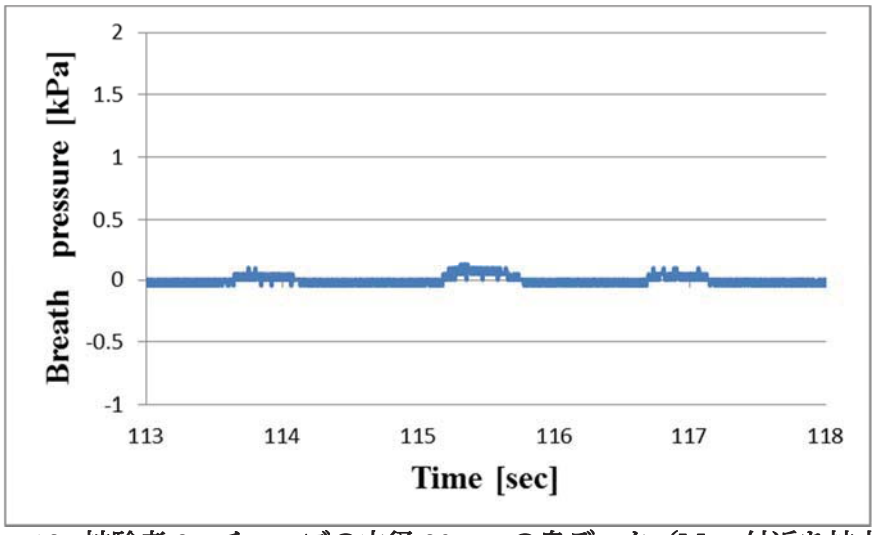
図ｖ.15: 被験者 3　チューブの内径 30mm の息データ（全体）



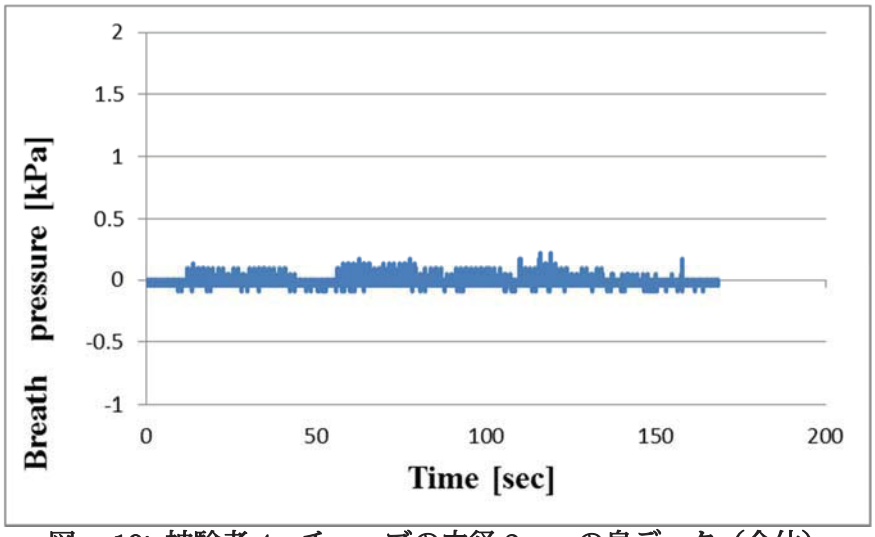
図ｖ.16: 被験者 3　チューブの内径 2mm の息データ（Max 付近を拡大）



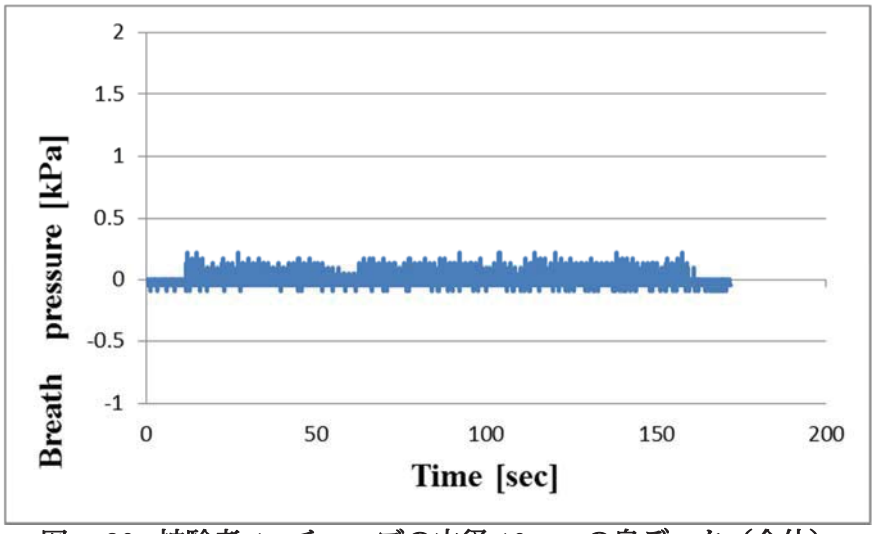
図ｖ.17: 被験者 3　チューブの内径 10mm の息データ（Max 付近を拡大）



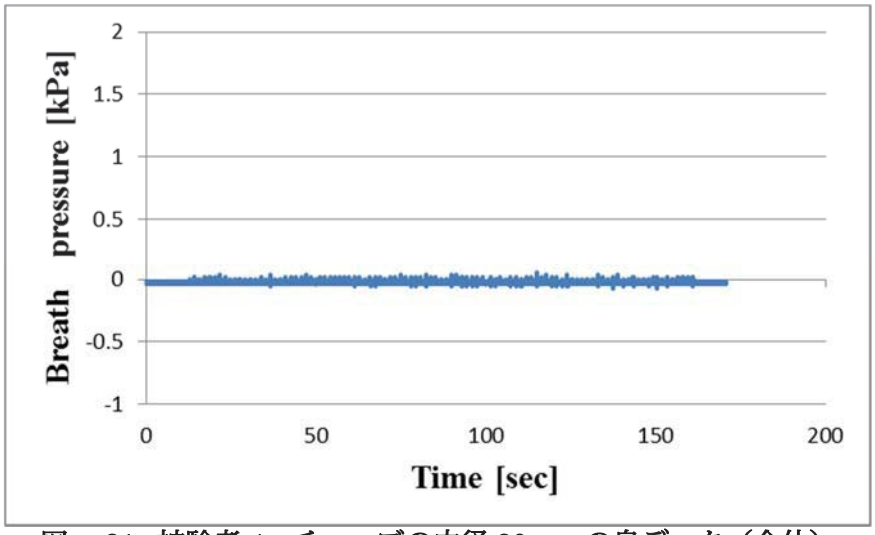
図ｖ.18: 被験者 3　チューブの内径 30mm の息データ（Max 付近を拡大）



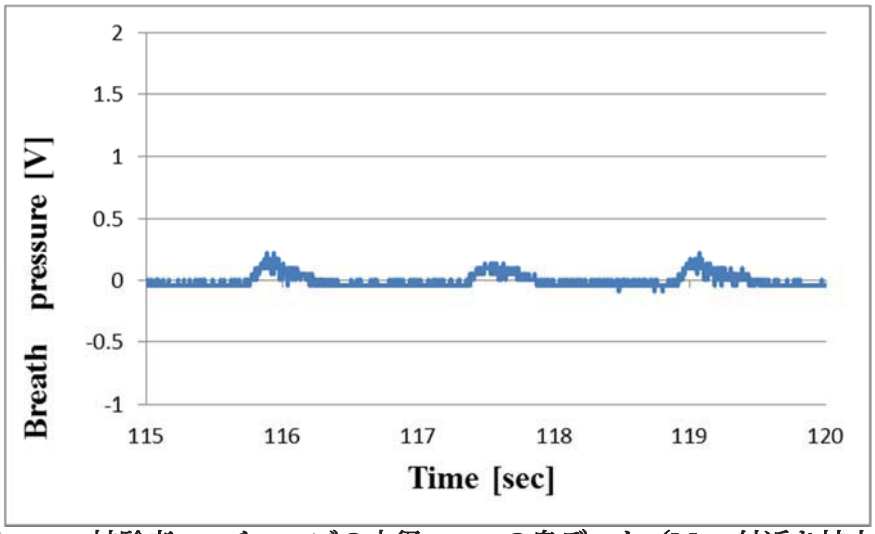
図ｖ.19: 被験者 4　チューブの内径 2mm の息データ（全体）



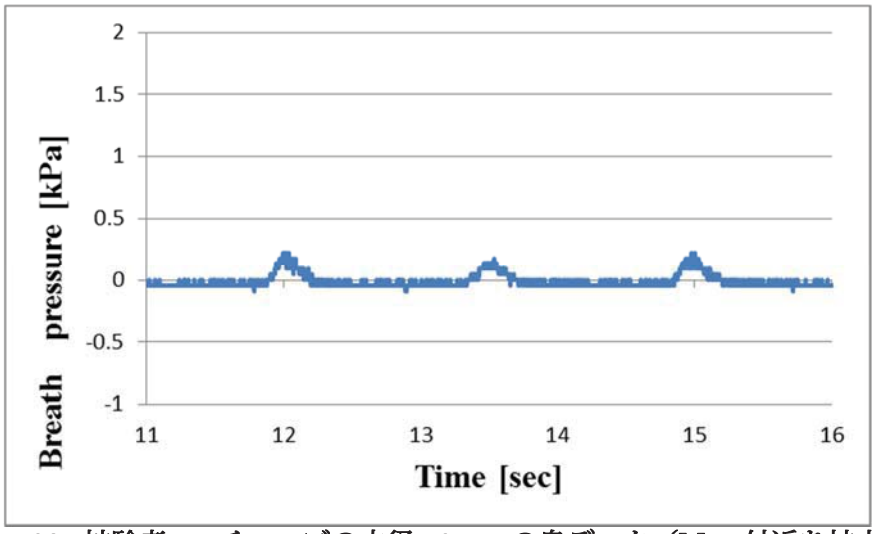
図ｖ.20: 被験者 4　チューブの内径 10mm の息データ（全体）



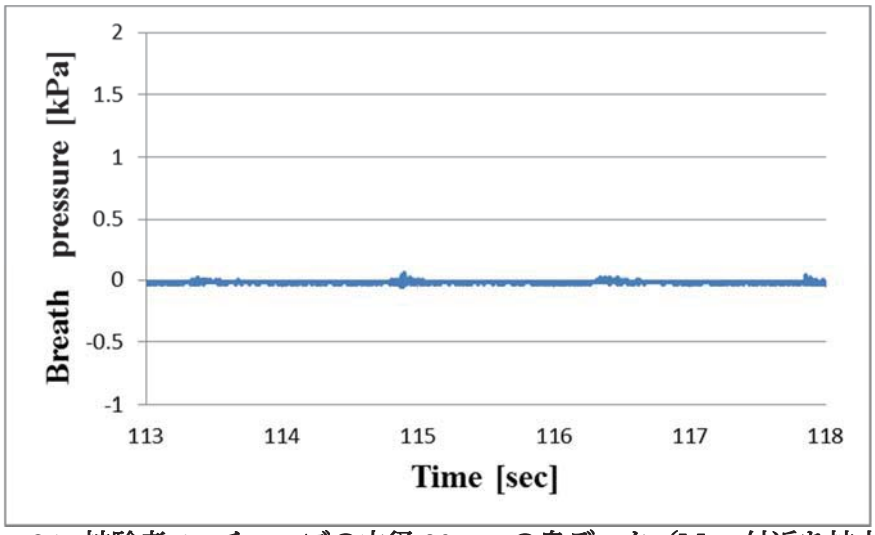
図ｖ.21: 被験者 4　チューブの内径 30mm の息データ（全体）



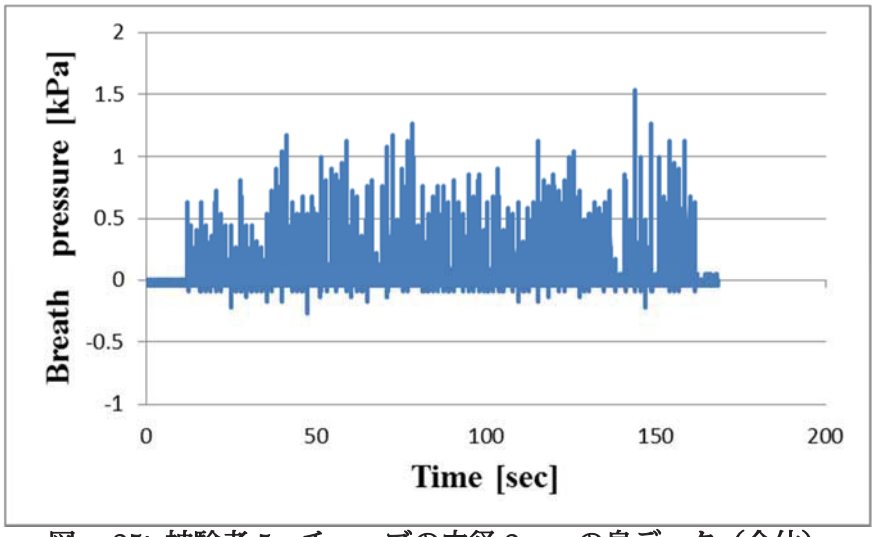
図ｖ.22: 被験者 4　チューブの内径 2mm の息データ（Max 付近を拡大）



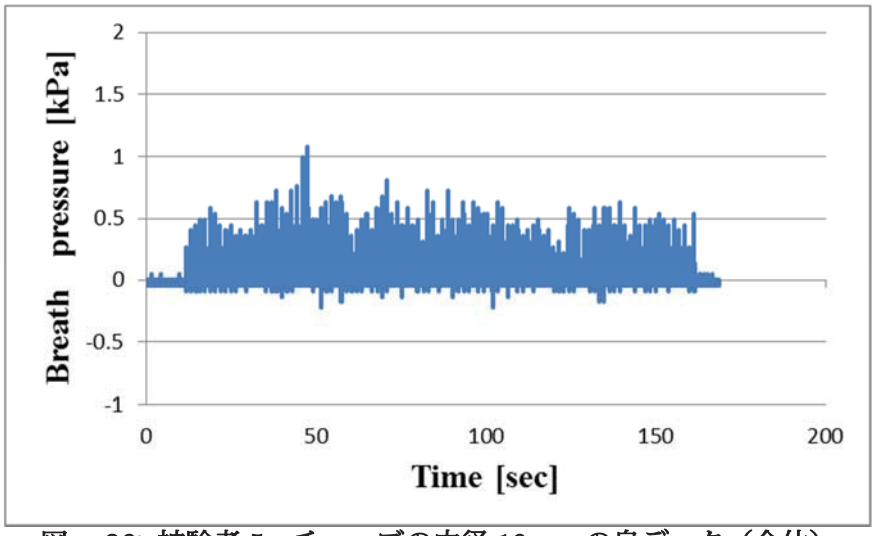
図ｖ.23: 被験者 4　チューブの内径 10mm の息データ（Max 付近を拡大）



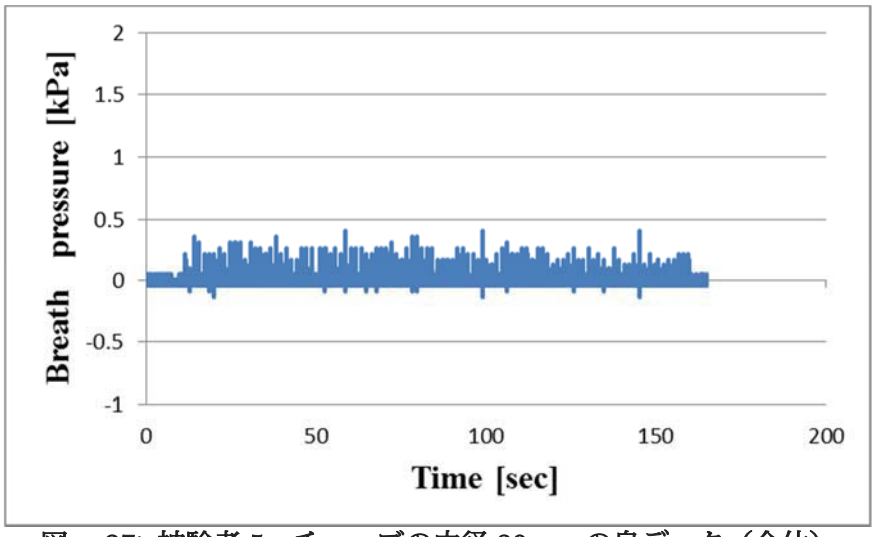
図ｖ.24: 被験者 4　チューブの内径 30mm の息データ（Max 付近を拡大）



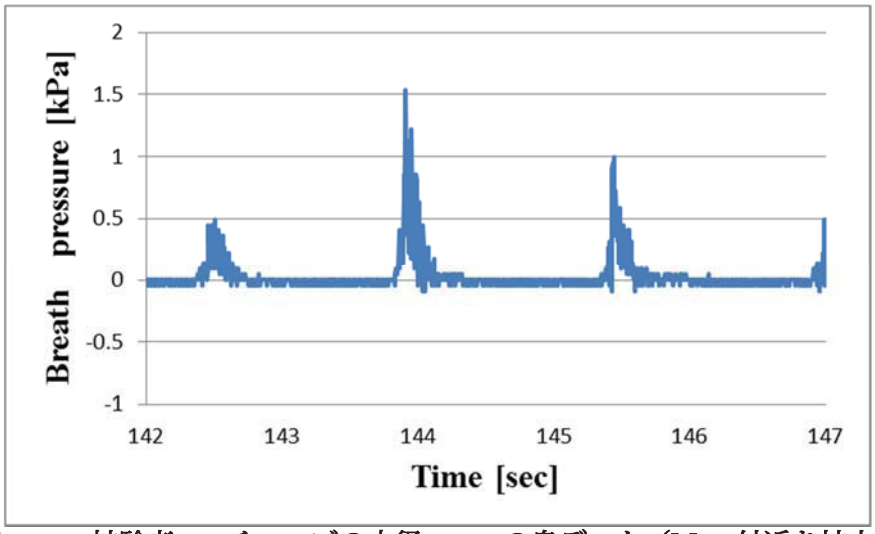
図ｖ.25: 被験者 5　チューブの内径 2mm の息データ（全体）



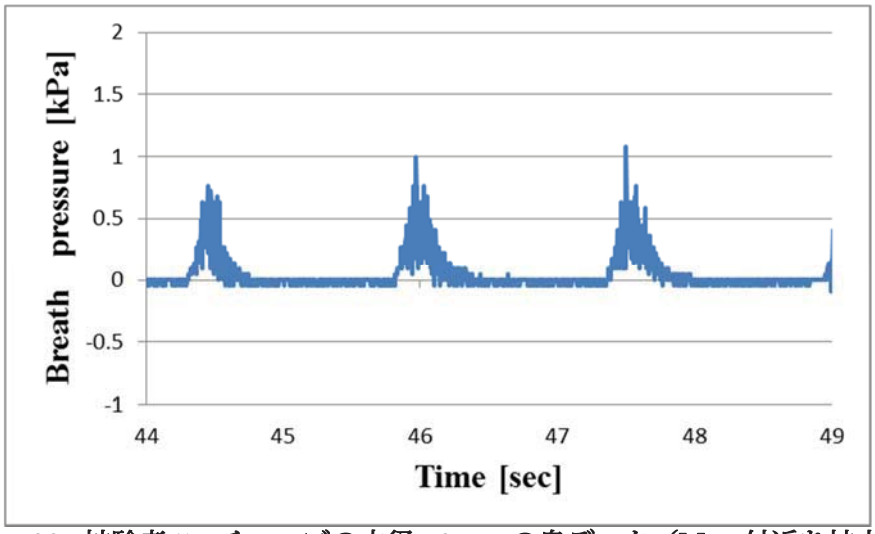
図ｖ.26: 被験者 5　チューブの内径 10mm の息データ（全体）



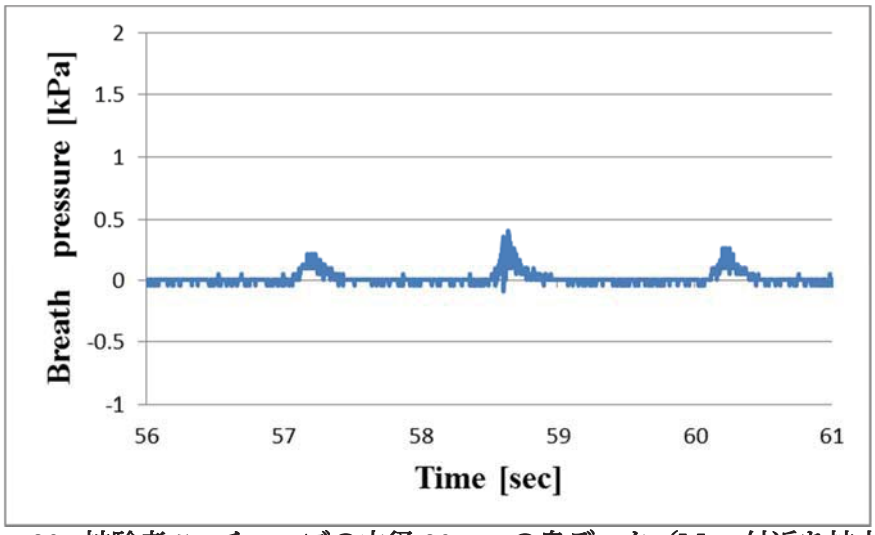
図ｖ.27: 被験者 5　チューブの内径 30mm の息データ（全体）



図ｖ.28: 被験者 5　チューブの内径 2mm の息データ（Max 付近を拡大）



図ｖ.29: 被験者 5　チューブの内径 10mm の息データ（Max 付近を拡大）



図ｖ.30: 被験者 5　チューブの内径 30mm の息データ（Max 付近を拡大）

VI 顔検出と口検出のチェックシート

被験者：＿＿＿＿＿＿＿＿

【 セット目】

領域	検出箇所	1回目	2回目	3回目	4回目	5回目	6回目	7回目	8回目	9回目	10回目	集計 ○	集計 △	集計 ×
① 無	顔													
	口													
② 右	顔													
	口													
③ 右連続	顔													
	口													
④ 左	顔													
	口													
⑤ 左連続	顔													
	口													
⑥ クリック	顔													
	口													
⑦ 上	顔													
	口													
⑧ 下	顔													
	口													

検出＝○、誤検出＝△、未検出＝×

Ⅶ 作業時間アンケート

被験者：

作業時間：　　分　　秒

作業時間：　　分　　秒

作業時間：　　分　　秒

【インターフェースの条件（3.2.1 節）】
以下の項目を 5 段階で評価し、あてはまるものに○をつけて下さい。
Q1-1：このインターフェースは被験者による操作を検知できますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q1-2：このインターフェースは安全ですか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q1-3：このインターフェースは清潔を保てますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q1-4：このインターフェースは直感的に操作ができますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q1-5：このインターフェースは患者のプライバシーが守られますか？※1
※1　操作内容を外部へ漏らさない。

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

【操作盤の条件（5.4.1 節）】
以下の項目を 5 段階で評価し、あてはまるものに○をつけて下さい。
Q2-1：操作盤は、ひと目で操作の内容がわかりますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q2-2：操作盤は、操作の邪魔にならないですか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q2-3：人の動きと操作がかみあっていますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

【ユニバーサルデザイン 7 原則】
以下の項目を 5 段階で評価し、あてはまるものに○をつけて下さい。
Q3-1：使う人が誰であろうと、公平に操作できると思いますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q3-4：操作時に自分のやり方、自分のペースで使用できますか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q3-6：身体への負担は、少ないですか？

全くそうは思わない ← 1 2 3 4 5 → 十分にそう思う

Q4-1：Q1～Q3 にて、1～2 に丸をつけた方への質問です。それは、なぜですか？

質問番号 Q -
理由：

質問番号 Q -
理由：

質問番号 Q -
理由：

Q4-2：ご意見、ご感想をお聞かせ下さい。

アンケートは以上です。ご協力ありがとうございました。