修士学位論文

論文題目　ジェスチャーによる
車載機器の操作

ふりがな　ひらやま　ともゆき
氏名　平山　智之

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）　2016年3月

工学院大学大学院

論文要旨

近年の自動車内にはカーナビゲーションシステムやカーオーディオシステムをはじめとした様々な車載機器があり，多くのドライバーがこれらの車載機器を利用している．車載機器の操作方法は，運転席と助手席の間の前方中央部にある操作パネルまたは画面を直接手で触れて操作するのが一般的であり，これが原因の脇見運転による事故が多く起きている．これを減らすことが本研究の目指すところである．

そのために，本研究ではオーディオなどの車載機器を操作する際，画面や操作パネルを見ずにジェスチャーによって操作が可能なインターフェイスを作製する．まず，自動車内での設置位置を決めるために認識率を求める実験を自動車内で行った．その結果，車内前方中央のシフトレバー付近に設置するのが良いことがわかった．また，その実験でジェスチャーの認識率が低いことが明らかになったので，ジェスチャーの認識パラメータの調整を行い，認識率の向上に成功した．最後に，認識パラメータ以外の要因が認識率に与える影響についても考察した．

目次

論文要旨 1
目次 2
第 1 章　序論 4
1.1.　研究背景 4
1.2.　関連技術 6
1.2.1.　車載機器の操作 6
1.2.2.　ジェスチャーによる操作（Ⅰ） 7
1.2.3.　ジェスチャーによる操作（Ⅱ） 8
1.3.　本研究の目的 9
1.4.　本論文の構成 10
第 2 章　システムの概要 11
2.1.　目的 11
2.2.　Leap Motion 12
2.2.1.　Kinect との比較 14
2.2.2.　ジェスチャーの種類 15
2.3.　仕様 17
第 3 章　プログラム 18
3.1.　ジェスチャーの有効化 18
3.2.　フレーム 19
3.3.　ジェスチャー判別 21
3.3.1.　タイミング 21
3.3.2.　スワイプ方向の判別 24
3.3.3.　サークル方向の判別 25
3.3.4.　処理 26
3.4.　オーディオプレイヤー 28

3.5.　プログラムの改良 32
第 4 章　認識率実験 34
4.1.　実験方法 35
4.2.　結果及び考察 36
4.2.1.　認識率 36
4.2.2.　誤認識 39
第 5 章　認識率の向上 43
5.1.　認識パラメータ調整 43
5.2.　パラメータ調整後認識率 45
5.2.1.　実験方法 45
5.2.2.　結果及び考察 45
5.3.　認識率に影響する要因 49
5.3.1.　設置角度による認識率の比較 50
5.3.2.　PC の違いによる認識率の比較 52
5.3.3.　フレーム数の表示 54
第 6 章　結論 55
第 7 章　謝辞 57
参考文献 58
図表目次 59
付録 61

第1章　　序論

1.1.　研究背景

今日の日本において，自動車はなくてはならない存在である．

しかし，表 1.1 によると交通事故の件数が年間約 54 万件発生していると言う問題がある．その中で脇見運転が原因の事故が全体の 16.8%の 9.1 万件発生している．これは，安全不確認の 30.6%に次いで 2 番目に多い原因となっている．

脇見運転の要因は大きく

- 運転に関係のない操作や動作
- 風景や看板など周囲への注視
- 荷物の落下など車内の急な変化への対応

の 3 つに分けられる．この中でも「運転に関係のない操作や動作」が約半数の 47.4%を占めている[2]．この「運転に関係のない操作や動作」にはカーオーディオやカーナビゲーションシステム，エアコンの操作や地図の閲覧、スマートフォンの操作などがある．

近年の自動車内にはカーナビゲーションシステムやカーオーディオシステムをはじめとした様々な車載機器があり，多くのドライバーがこれら車載機器を利用している．車載機器の操作方法は，運転席と助手席の間の前方中央部にある操作パネルまたは画面を直接手で触れて操作するのが一般的であり，これが上で見たように事故の一因となっている．

以上のように自動車をより便利で快適なものにするためのカーナビやオーディオが原因の事故を減らすことが本研究の目標である．

表 1.1　原付以上運転者の法令違反別交通事故件数（平成 26 年中）　[1]

法令違反別	件数	構成率
信号無視	15, 702	2. 9%
通行区分	3, 467	0. 6%
最高速度	1, 096	0. 2%
横断・転回等	4, 292	0. 8%
追越し	1, 408	0. 3%
踏切不停止	30	0. 0%
右折違反	1, 663	0. 3%
左折違反	3, 745	0. 7%
優先通行妨害	11, 918	2. 2%
交差点安全進行	31, 516	5. 8%
歩行者妨害等	13, 197	2. 4%
徐行	5, 851	1. 1%

法令違反別		件数	構成率
一時不停止		23, 091	4. 2%
酒酔い運転		213	0. 0%
過労運転		378	0. 1%
安全運転義務	転操作不適	38, 133	7. 0%
	漫然運転	43, 600	8. 0%
	脇見運転	91, 293	16. 8%
	動静不注視	62, 240	11. 4%
	安全不確認	166, 450	30. 6%
	安全速度	4, 635	0. 9%
	その他	6, 831	1. 3%
その他の違反		12, 637	2. 3%
違反不明		893	0. 2%
合計		544, 279	100%

1.2. 関連技術

1.2.1. 車載機器の操作

現在，操作パネルを利用せずに車載機器の操作を行う方法の一つとして，音声認識がある．

音声認識では音声によって操作するため，進行方向から視線を逸らすことなく操作が可能である．

自動車で音声認識をするためには，自動車内に図 1.1 のような音声認識マイクを取り付け，スイッチを押してから発話することで認識されるようになる．

一般に音声認識マイクはハンドル後方か，ルームミラー付近の天井や窓に取り付けられていることが多く，運転手の口元から 40～50cm 程離れてしまう．しかし，走行中の自動車内には，エンジン音や走行音の他に風切り音やエアコン，ワイパーやウィンカーの音，ラジオやカーステレオなどのオーディオ類，同乗者の声など多くの雑音や騒音が存在する．そのため，離れた位置にあるマイクで音声認識をすることは困難である．

また，どの様に言えば認識されるのかがわからず，戸惑ってしまうことや，うまく認識されず何度も同じ発話を求められ，飽きてしまうユーザーも多い[4,5]．

図 1.1 トヨタ社製音声認識マイク&スイッチ [3]

1.2.2. ジェスチャーによる操作（I）

操作パネルや画面に触れずに機器を操作する手段の一つにジェスチャー認識が挙げられる．車載機器以外をジェスチャーで操作する例として任天堂社のWii(図 1.2)や，マイクロソフト社の Kinect（図 1.3）などの家庭用ゲーム機のコントローラーが挙げられる．

Wiiでは，付属のWiiリモコンを手に持ってジェスチャーを行う必要がある．テレビ画面上部または下部にセンサーバーユニットを設置し，画面と Wii リモコンの位置関係を把握する．さらに Wii リモコン内部のジャイロセンサや加速度センサでリモコンの動きを検出することによってジェスチャーでの操作が可能となっている．

一方，Kinectは本体に距離センサや映像センサが内蔵されており，これを用いて人の骨格や動きをトラッキングできるため，Wii のように Wii リモコンを手に持つ必要がない．

図 1.2　左：センサーバーユニット，右：Wiiリモコンと本体　[6]

図 1.3　Kinect　[7]

1.2.3. ジェスチャーによる操作（Ⅱ）

1.2.2 では家庭用ゲーム機向けのジェスチャーデバイスを例に挙げたが，その他のジェスチャーによる操作が可能な機器として，3D モーションセンサ Leap Motion（図 1.4）が挙げられる．

Leap Motion は，PC の操作をマウスなどを使用せずに手の動きで入力することができる USB 型デバイスである．タッチパネル非対応のモニターを使用していても，Leap Motion によって Windows 8 のように指の操作で行うことができる．

また，日本 HP 社から Leap Motion が内蔵されたノート PC「HP ENVY 17-j100 Leap Motion SE」（図 1.5）も販売されている．

図 1.4　Leap Motion 本体

図 1.5　 HP ENVY 17-j100 Leap Motion SE　[8]

1.3. 本研究の目的

これまで見たように，カーオーディオやカーナビゲーションシステム，エアコンなど自動車内には多くの車載機器があるが，走行中にこれらを操作することによる脇見運転が原因の事故が多く起きている．

音声認識で操作可能な自動車も販売されているが，使いこなせるユーザーが少ないと言う現状がある．

また，近年のゲーム市場では，Wii や Kinect のようにボタン操作ではなく，ジェスチャーによる操作が広く普及し，Leap Motion のように PC を操作するためのジェスチャーデバイスも普及しはじめている．

そこで，本研究ではオーディオなどの車載機器を操作する際に，画面や操作パネルを見ずにジェスチャーによって操作が可能なインターフェイスを作製し，走行中に車載機器を操作することが原因の脇見運転による事故の減少を目指す．

1.4. 本論文の構成

本論文の構成を以下に示す.

第 2 章では本研究の準備段階として, システムの概要や Leap Motion について述べ, 利用するジェスチャーや仕様を示す.

第 3 章では作製したプログラムの流れを述べ, 改良を加えたプログラムの変更点を示す.

第 4 章では Leap Motion の設置位置によるジェスチャーの認識率の違いを実験により調べ，結果とジェスチャーの誤認識について述べる.

第 5 章では第 4 章での実験結果を元に，認識率を向上させ，誤認識を減らすために認識パラメータを調整した前後での結果を比較し考察する．また，認識パラメータ以外の観点から認識率に影響する要因について述べる.

第 6 章では本研究で得られた成果を要約し，結論とする.

第2章　　システムの概要

2.1.　目的

本研究の目的であるジェスチャーによる車載機器の操作を可能にするために，自動車内に 3D モーションセンサ Leap Motion（図 2.1）を設置する．本研究では Windows PC 上でオーディオ操作の動作検証を行うが，将来 Leap Motion の Android 用 SDK が公開された際に Android OS 搭載のカーナビゲーションシステムと接続できると考える．

図 2.1　Leap Motion 本体

2.2. Leap Motion

Leap Motion はアメリカの Leap Motion 社から販売されている手や指の検出に特化したセンサである．本体表面に可視光を通しにくく，赤外線を通すフィルタが取り付けられ，内部に 3 つの赤外線 LED とステレオカメラ（カメラ 2 基）で構成されている[9].

Leap Motion でできることは，手や指の検出の他に，棒状のもの（ツール）の検出や，これらの動きからのジェスチャーの認識などが挙げられる.

Leap Motion の動作環境を表 2.1 に示す.

表 2.1 Leap Motion の動作環境 [10]

項目	仕様
対応 OS	Windows 7/8/8.1, Mac OS X 10.7 以降
CPU	AMD PhenomTM II,Intel ® CoreTM i3 以上
メモリ	2GB RAM 以上
USB インターフェイス	USB 2.0 ポート

Leap Motion で取得されるデータは右手系のデカルト座標系で表現されており（図 2.2），Leap Motion の中心を基点として，そこからの距離が mm 単位で取得される. Leap Motion の右側，上側，手前側がそれぞれ X,Y,Z 座標の正の向きである.

また，図 2.3 に示すインタラクションボックス内が Leap Motion の認識範囲である[11].

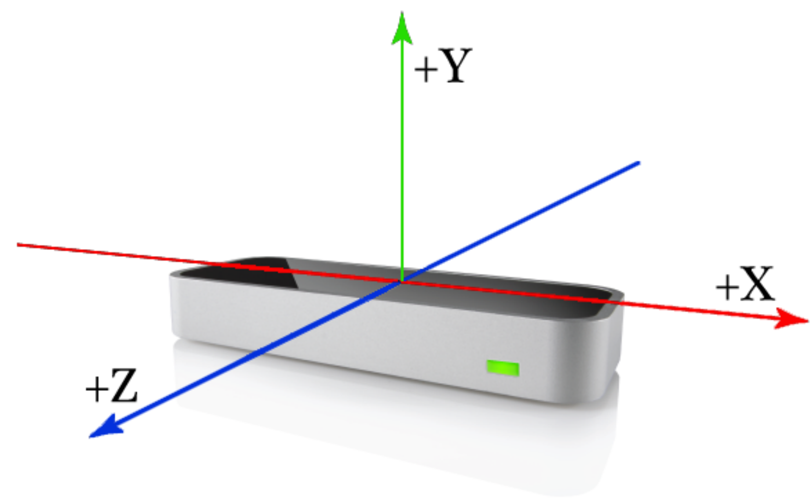

図 2.2　Leap Motion の座標系　[11]

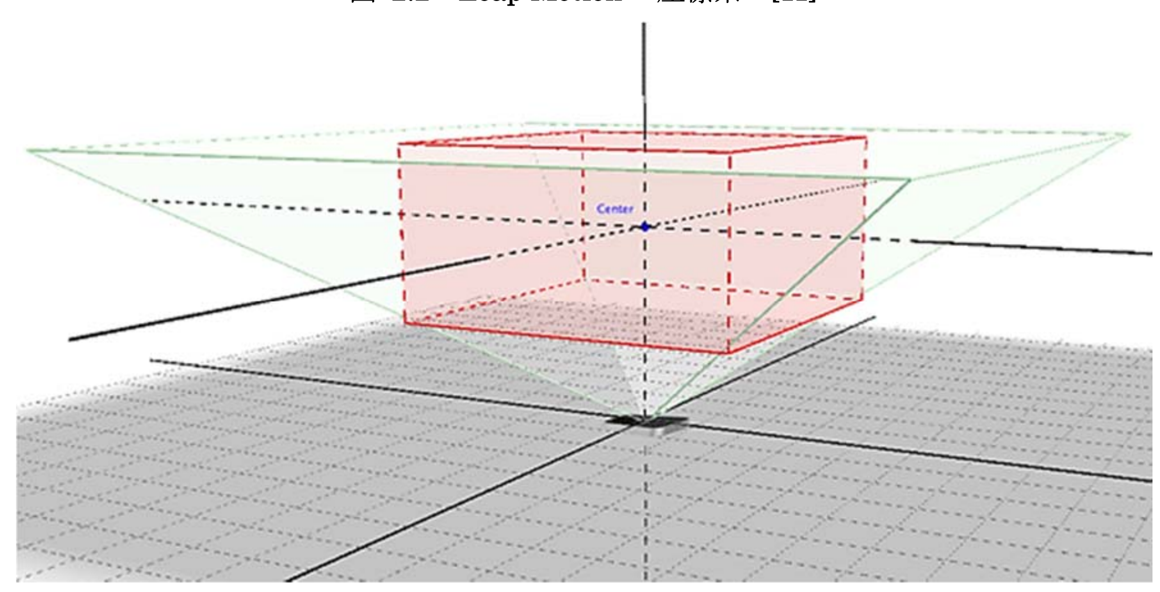

図 2.3　インタラクションボックス　[11]

2.2.1. Kinectとの比較

Leap Motion と 1.2.2 で紹介した Kinect との仕様の比較を表 2.2 に示す．

Kinect は全身や上半身の骨格や顔を認識，トラッキングすることができるモーションセンサデバイスである．しかし，指やその動きを検出することは出来ない．また，寸法が 280x65x70[mm]とやや大きく，距離センサの認識範囲が近距離モードでも 400～3,000[mm]となっている．

一方，Leap Motion は全身を認識，検出することは出来ないが，手や指に特化しこれらの細かな動きも認識することが出来る．寸法は 80x30x11[mm]と消しゴムくらいの大きさで，認識範囲は 25～600[mm]と Kinect に比べ，近い距離での認識が可能となっている．

狭い自動車内での利用を考えると，手指のみの検出となるが，より小型で認識範囲も近距離である Leap Motion の方が適していると言える．

表 2.2 Leap Motion と Kinect の比較 [7,10]

	Leap Motion	Kinect for Windows
寸法	80(W)x30(D)x11(H)[mm]	280(W)x65(D)x70(H) [mm]
重量	45[g]	600[g]
認識距離	25～600[mm]	（近距離モード）400～3000[mm]
検出部位	手・指	全身・上半身・顔
音声認識	不可	可

2.2.2. ジェスチャーの種類

本節では本研究で主に用いる Leap Motion のジェスチャー認識機能について紹介する.

以下に Leap Motion でサポートされているジェスチャーの種類を示す.

- スワイプジェスチャー

図 2.4 のように指の直線的な，すばやい動きをスワイプジェスチャーとして認識できる.

前後方向や上下左右方向，どの方向でも認識することができる.

ジェスチャーが終わるまで検出をし続けるが，指の移動方向が変わるか，動きが遅くなるとスワイプジェスチャーの認識を終了する.

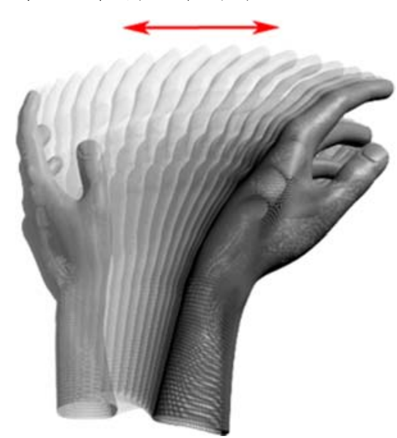

図 2.4 スワイプジェスチャー [11]

- サークルジェスチャー

図 2.5 のように指先で円を描くような動きをサークルジェスチャーとして認識できる.

サークルジェスチャーには時計回りと反時計回りの 2 つの向きがある.

指の動きが円ではなくなるか，遅くなるとサークルジェスチャーの認識を終了する.

図 2.5 サークルジェスチャー [11]

- キータップジェスチャー

図 2.6 のようにキーボードを押すように下向きにすばやく指を弾くように動かすことでキータップジェスチャーを認識できる．

図 2.6　キータップジェスチャー ［11］

- スクリーンタップジェスチャー

図 2.7 のように垂直なタッチスクリーンに指を触れるように，前向きにたたくことでスクリーンタップジェスチャーを認識できる．

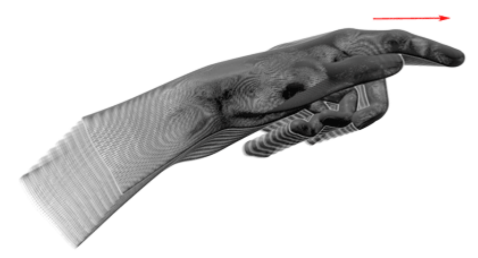

図 2.7　スクリーンタップジェスチャー ［11］

キータップ，スクリーンタップジェスチャーはスワイプ，サークルジェスチャーと異なり単発的に認識する．

2.3.　仕様

本研究では車載機器の中でもオーディオの操作に特化したジェスチャー操作システムを開発する．

その理由として，カーナビでは地図の操作や目的地の検索や指定などが必要となり，画面を見ることが操作の前提にあることが挙げられる．よって，走行中の脇見運転による事故を減少させると言う本研究の目的には適していない．

また，エアコンの操作では自動車のシステム内部に手を加える必要があり，自動車での再現，検証が困難であるため今回は見送った．

オーディオ操作時には「曲の再生及び一時停止」，「曲の停止」，「前の曲に戻る」，「次の曲に移る」，「音量の調節（大，小 2 種類）」の少なくとも 6 種類のジェスチャーが必要になると考えた．2.2.2 で述べたように Leap Motion では 4 種類のジェスチャーが用意されているが，キータップジェスチャーとスクリーンタップジェスチャーの 2 種類は，他のスワイプジェスチャー，サークルジェスチャーなどと同時に利用するとジェスチャーの認識が困難であり，ジェスチャーの難易度が上がるため，本研究ではスワイプジェスチャーとサークルジェスチャーの 2 種類を利用する．また，スワイプジェスチャーは前後方向を除いた上下左右方向の計 4 方向，サークルジェスチャーは時計回りと反時計回りの 2 方向の計 6 種類のジェスチャーが利用できる．

オーディオ操作のジェスチャーの割り当ては，表 2.3 に示すように定める．一般的な AV 機器と同じような左右や回転の動作を用いることで，ユーザーが直感的に操作を行えるようになると考えられる．

表 2.3　操作割り当て

操作項目	ジェスチャー
再生	上スワイプ ↑
一時停止	上スワイプ ↑
停止	下スワイプ ↓
次の曲に進む	右スワイプ →
前の局に戻る	左スワイプ ←
音量上げる	時計回りサークル ↻
音量下げる	反時計回りサークル ↺

第3章　　プログラム

ジェスチャーでオーディオ操作を行うアプリケーションを作製するにあたり，まずはコンソール画面に認識したジェスチャーを表示するプログラムを C++，C#のそれぞれの言語で作製した．以下，そのプログラムについて解説する．

3.1.　ジェスチャーの有効化

ジェスチャーを利用するには，各ジェスチャーの有効化を宣言しなければならない．

C++では main 関数内で enableGesture()によって，C#では OnConnect 関数内で EnableGesture()によってジェスチャーの有効化を宣言しそれぞれのジェスチャーを有効化できる．

- TYPE_CIRCLE:サークルジェスチャー
- TYPE_KEY_TAP：キータップジェスチャー
- TYPE_SCREEN_TAP：スクリーンタップジェスチャー
- TYPE_SWIPE:スワイプジェスチャーの

2.3 で述べたように本研究ではキータップ，スクリーンタップジェスチャーは使用しないため，以下のようにサークル，スワイプジェスチャーのみ有効化させる．

C++

```
// SWIPE と CIRCLE ジェスチャーを有効にする
leap.enableGesture( Leap::Gesture::Type::TYPE_SWIPE );
leap.enableGesture( Leap::Gesture::Type::TYPE_CIRCLE );
```

C#

```
// SWIPEとCIRCLEジェスチャーを有効にする
controller.EnableGesture(Gesture.GestureType.TYPE_SWIPE);
controller.EnableGesture(Gesture.GestureType.TYPE_CIRCLE);
```

3.2. フレーム

Leap Motion ではパラパラ漫画の一コマのようなフレームがデータの中心になっている．フレームにはその瞬間の「Leap Motion の状態」，「検出した手や指」，「認識したジェスチャー」などの情報がまとめられている．これを連続的に処理することでアプリケーションの動作を定めることができる．

まず，最初の処理を行うための現在のフレームと以前のフレームとなる変数を宣言する．

C++
```
bool mFirst;
Leap::Frame mCurrentFrame; // 現在のフレーム
Leap::Frame mLastFrame; // 以前のフレーム
```

C#
```
bool mFirst;
Frame mCurrentFrame; // 現在のフレーム
Frame mLastFrame; // 以前のフレーム
```

leap.frame()を実行することで，最新のフレームが取得できる．引数としては 0 から 59 までの値を指定でき，最新のフレームから 59 フレーム前までの計 60 フレームを取得することができるが，何も指定していない場合は最新のフレームが取得される．

以下のようにプログラム起動時に一度だけ mCurrentFrame に現在のフレームを格納し，以後この処理は行わない．そして mLastFrame に mCurrentFrame を格納し，mCurrentFrame には新たに現在のフレームを格納し，フレームを更新している．

C++
```
if(mFirst){
    mCurrentFrame = leap.frame();
    mFirst = false;
}
// フレームの更新
    mLastFrame = mCurrentFrame;
    mCurrentFrame = leap.frame();
```

C#

```
public override void OnFrame(Controller controller)
{
    if (mFirst)    {
        mCurrentFrame = controller.Frame();
        mFirst = false;
    }
    // フレームの更新
    mLastFrame = mCurrentFrame;
    mCurrentFrame = controller.Frame();
```

3.3. ジェスチャー判別

3.3.1. タイミング

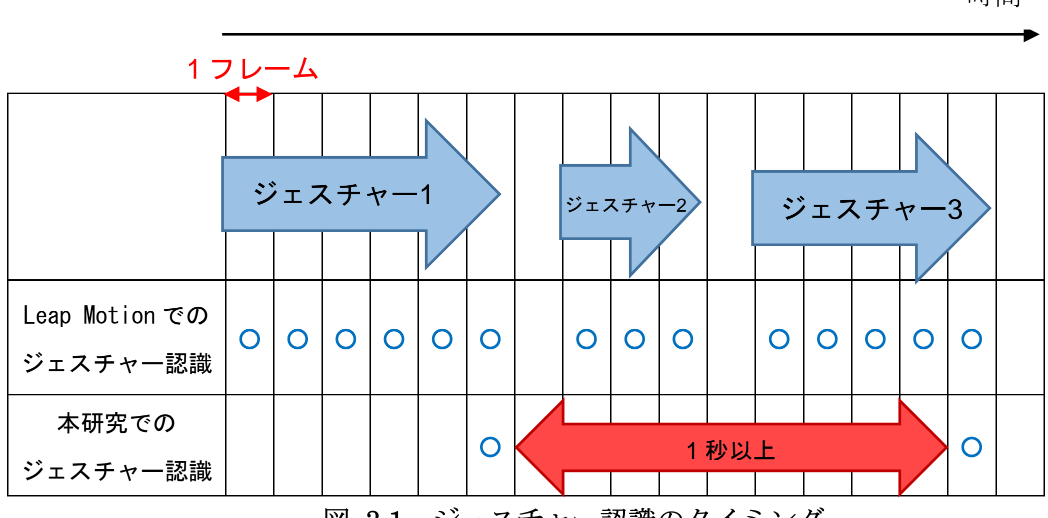

図 3.1　ジェスチャー認識のタイミング

本研究ではジェスチャーを認識した際，以前のジェスチャーより 1 秒以上経過しており，なおかつそれがジェスチャーの停止あらわすときにオーディオ操作を行う(図 3.1)．そのためには，ジェスチャーを以前に認識した時刻を保存しておく変数が必要なため，以下のように mPrevGestureTime を宣言し，初期値を 0 とする．

C++

```
int64_t mPrevGestureTime; // 以前にジェスチャーを認識した時刻

// 変数初期化用コンストラクタ
SampleListener(){
    mFirst = true;
    mPrevGestureTime = 0; // 以前にジェスチャーを認識した時刻
}
```

C#

```
long mPrevGesureTime; // 以前にジェスチャーを認識した時刻

public SampleListener()
{
    mFirst = true;
    mSwipeGestureList = new List<SwipeGesture>();
    mCircleGestureList = new List<CircleGesture>();

    mPrevGestureTime = 0;
}
```

次に Leap Motion によってジェスチャーが認識された際に呼ばれるイベントハンドラ onFrame 内でいくつかの変数を定義する．ジェスチャーが停止状態かどうかを格納する変数「(ジェスチャー名）Stopped」や，ジェスチャーを認識した時刻を格納する変数「(ジェスチャー名）Time」である．

既に述べたように，オーディオ操作を連続で実行しないための待機時間を detectionWaitTime と宣言し，その時間を 1,000,000[μ 秒]（=1.0[秒]）とする．

C++

```
bool leftSwipeStopped = false;
bool rightSwipeStopped = false;
bool upSwipeStopped = false;
bool downSwipeStopped = false;
bool clockwiseCircleStopped = false;
bool coclockwiseCircleStopped = false;

int64_t leftSwipeTime; // 左スワイプの認識時刻
int64_t rightSwipeTime; // 右スワイプの認識時刻
int64_t upSwipeTime; // 上スワイプの認識時刻
int64_t downSwipeTime; // 下スワイプの認識時刻
int64_t clockwiseCircleTime; // 時計回りサークルを認識した時刻
int64_t coclockwiseCircleTime; // 反時計回りサークルを認識した時刻

int64_t detectionWaitTime =  1000000; // 1000*1000 (μ秒) = 1秒。これ以内の短い間隔の
ジェスチャーは認識しない。
```

C#

```
bool leftSwipeStopped = false;
bool rightSwipeStopped = false;
bool upSwipeStopped = false;
bool downSwipeStopped = false;
long leftSwipeTime = 0; // 左スワイプの認識時刻
long rightSwipeTime = 0; // 右スワイプの認識時刻
long upSwipeTime = 0; // 上スワイプの認識時刻
long downSwipeTime = 0; // 下スワイプの認識時刻

bool clockwiseStopped = false;
bool counterClockwiseStopped = false;
long clockwiseTime = 0; // 時計回りサークルの認識時刻
long counterClockwiseTime = 0; // 反時計回りサークルの認識時刻

long detectionWaitTime = 1000000; // 1000*1000 (μ秒) = 1秒。これ以内の短い間隔のジェス
チャーは認識しない。
```

ここからは左スワイプを例に示す．

以下のように gesture.state()でスワイプジェスチャー状態を取得する．それが停止状態であり，なおかつ向きが左であれば我々の基準での左スワイプ認識の候補であるとし，leftSwipeStopped と leftSwaipeTime に値を設定する．

C++
```
if( gesture.state()==Leap::Gesture::State::STATE_STOP){ // スワイプ停止の検出
    if(gesture.direction().x<=-0.5){  // 左スワイプの認識
        leftSwipeStopped = true;
        leftSwipeTime = gesture.frame().timestamp();
       }
}
```

C#
```
if (gesture.State == Leap.Gesture.GestureState.STATE_STOP){ // スワイプ停止の検出
    if (gesture.Direction.x <= -0.5){  // 左スワイプの認識
        leftSwipeStopped = true;
        leftSwipeTime = gesture.Frame.Timestamp;
    }
}
```

さらに，左スワイプを認識した際の時間と，以前にジェスチャーを認識した時間（オーディオ操作をした時間）との間隔が 1 秒より長ければ我々の基準での左スワイプを認識し，処理 1 を行う．

その後，処理 2 として，mPrevGestureTime に左スワイプを認識した時刻を格納する．

C++・C#共通
```
if(leftSwipeStopped){ // 左スワイプが認識されたら
    if(leftSwipeTime - mPrevGestureTime > detectionWaitTime){ // 以前のジェスチャーよ
        り1秒以上たっていたら

            （処理1）

            mPrevGestureTime = leftSwipeTime; // 処理2
    }
}
```

3.3.2. スワイプ方向の判別

さきほどは例として左スワイプの認識のみで解説したが他のジェスチャーも含めたコードでは以下のようになっている．

ジェスチャーの向きを表すベクトル(x,y)の値による判定を行っていることがわかる．

C++

```
for ( auto gesture : mSwipeGestureList ) {
    if( gesture.state()==Leap::Gesture::State::STATE_STOP) { // スワイプ停止の検出
        if(gesture.direction().x<=-0.5) {  // 左スワイプの認識
            leftSwipeStopped = true;
            leftSwipeTime = gesture.frame().timestamp();
        }else if(gesture.direction().x>=0.5) {  //右スワイプの認識
            rightSwipeStopped = true;
            rightSwipeTime = gesture.frame().timestamp();
        }
        if(gesture.direction().y>=0.5) {  // 上スワイプの認識
            upSwipeStopped = true;
            upSwipeTime = gesture.frame().timestamp();
        }else if(gesture.direction().y<=-0.5) {  //下スワイプの認識
            downSwipeStopped = true;
            downSwipeTime = gesture.frame().timestamp();
        }
    }
}
```

C#

```
foreach (SwipeGesture gesture in mSwipeGestureList) {
    if (gesture.State == Leap.Gesture.GestureState.STATE_STOP) { // スワイプ停止の検出
        if (gesture.Direction.x <= -0.5) {  // 左スワイプの認識
            leftSwipeStopped = true;
            leftSwipeTime = gesture.Frame.Timestamp;
        }else if (gesture.Direction.x >= 0.5) {  //右スワイプの認識
            rightSwipeStopped = true;
            rightSwipeTime = gesture.Frame.Timestamp;
        }
        if (gesture.Direction.y >= 0.5) {  // 上スワイプの認識
            upSwipeStopped = true;
            upSwipeTime = gesture.Frame.Timestamp;
        }else if (gesture.Direction.y <= -0.5) {  //下スワイプの認識
            downSwipeStopped = true;
            downSwipeTime = gesture.Frame.Timestamp;
        }
    }
}
```

3.3.3. サークル方向の判別

スワイプジェスチャーと同様にサークルジェスチャーの認識も行う．

サークルジェスチャーの方向の判別には，指で描いた円に対する法線ベクトルを C++では gesuture.nomal()，C#では gesture.Normal で取得する．時計回りの円を描いた場合､法線ベクトルは画面の奥側に，反時計回りに円を描くと法線ベクトルは画面の手前側を指す．指は画面の奥側を指しているので，法線ベクトルと指との角度が 90 度以下のとき，時計回りのサークルジェスチャーと認識される．

C++

```
for ( auto gesture : mCircleGestureList ) {
    if( gesture.state()==Leap::Gesture::State::STATE_STOP){ // サークル停止の検出
        if(gesture.pointable().direction().angleTo(gesture.normal()) <= (Leap::PI/2)){//
時計回りサークルの認識
            clockwiseCircleStopped = true;
            clockwiseCircleTime = gesture.frame().timestamp();
        }else {  //反時計回りサークルの認識
            coclockwiseCircleStopped = true;
            coclockwiseCircleTime = gesture.frame().timestamp();
        }
    }
}
```

C#

```
foreach (CircleGesture gesture in mCircleGestureList){
    if (gesture.State == Leap.Gesture.GestureState.STATE_STOP){ // サークル停止の検出
        if (gesture.Pointable.Direction.AngleTo(gesture.Normal) <= Math.PI / 2){// 時計回
りサークルの認識
            clockwiseStopped = true;
            clockwiseTime = gesture.Frame.Timestamp;
        }else{  //反時計回りサークルの認識
            counterClockwiseStopped = true;
            counterClockwiseTime = gesture.Frame.Timestamp;
        }
    }
}
```

3.3.4. 処理

3.3.1 でも述べた，我々の基準でのジェスチャーが認識されたときに実行する処理 1 では下記のようにコンソール画面に文字を表示する．

C++

```
if(leftSwipeStopped){ // 左スワイプが認識されたら
   if(leftSwipeTime - mPrevGestureTime > detectionWaitTime){ // 以前の左スワイプ認識より1秒以上たっていたら
      std::cout << "←  左スワイプ  ←" << std::endl; // 処理1
      mPrevGestureTime = leftSwipeTime; // 処理2
   }
}
if(rightSwipeStopped){ // 右スワイプが認識されたら
   if(rightSwipeTime - mPrevGestureTime > detectionWaitTime){ // 以前の右スワイプ認識より1秒以上たっていたら
      std::cout << "→  右スワイプ  →" << std::endl; // 処理1
      mPrevGestureTime = rightSwipeTime; // 処理2
   }
}
if(upSwipeStopped){ // 上スワイプが認識されたら
   if(upSwipeTime - mPrevGestureTime > detectionWaitTime){ // 以前の上スワイプ認識より1秒以上たっていたら
      std::cout << "↑  上スワイプ  ↑" << std::endl; // 処理1
      mPrevGestureTime = upSwipeTime; // 処理2
   }
}
if(downSwipeStopped){ // 下スワイプが認識されたら
   if(downSwipeTime - mPrevGestureTime > detectionWaitTime){ // 以前の下スワイプ認識より1秒以上たっていたら
      std::cout << "↓  下スワイプ  ↓" << std::endl; // 処理1
      mPrevGestureTime = downSwipeTime; // 処理2
   }
}
if(clockwiseCircleStopped){ // 時計回りサークルが認識されたら
   if(clockwiseCircleTime - mPrevGestureTime > detectionWaitTime){ // 時計回りサークル認識より1秒以上たっていたら
      std::cout << "○  時計回りサークル  ○" << std::endl; // 処理1
      mPrevGestureTime = clockwiseCircleTime; // 処理2
   }
}
if(coclockwiseCircleStopped){ // 反時計回りサークルが認識されたら
   if(coclockwiseCircleTime - mPrevGestureTime > detectionWaitTime){ // 時計回りサークル認識より1秒以上たっていたら
      std::cout << "◎  反時計回りサークル  ◎" << std::endl; // 処理1
      mPrevGestureTime = coclockwiseCircleTime; // 処理2
   }
}
```

```
C#
if (leftSwipeStopped) { // 左スワイプが認識されたら
    if (leftSwipeTime - mPrevGestureTime > detectionWaitTime) {  // 以前の左スワイプ認識より
1秒以上たっていたら
        SafeWriteLine("左スワイプ認識");
        mPrevGestureTime = leftSwipeTime;
    }
}
if (rightSwipeStopped) { // 右スワイプが認識されたら
    if (rightSwipeTime - mPrevGestureTime > detectionWaitTime) {  // 以前の右スワイプ認識よ
り1秒以上たっていたら
        SafeWriteLine("右スワイプ認識");
        mPrevGestureTime = rightSwipeTime;
    }
}
if (upSwipeStopped) { // 上スワイプが認識されたら
    if (upSwipeTime - mPrevGestureTime > detectionWaitTime) {  // 以前の上スワイプ認識より1
秒以上たっていたら
        SafeWriteLine("上スワイプ認識");
        mPrevGestureTime = upSwipeTime;
    }
}
if (downSwipeStopped) { // 下スワイプが認識されたら
    if (downSwipeTime - mPrevGestureTime > detectionWaitTime) {  // 以前の下スワイプ認識より
1秒以上たっていたら
        SafeWriteLine("下スワイプ認識");
        mPrevGestureTime = downSwipeTime;
    }
}
if (clockwiseStopped) { // 時計回りがが認識されたら
    if (clockwiseTime - mPrevGestureTime > detectionWaitTime) {  // 以前の時計回り認識より1
秒以上たっていたら
        SafeWriteLine("時計回り認識");
        mPrevGestureTime = clockwiseTime;
    }
}
if (counterClockwiseStopped) { // 時計回りがが認識されたら
    if (counterClockwiseTime - mPrevGestureTime > detectionWaitTime) {  // 以前の反時計回り
認識より1秒以上たっていたら
        SafeWriteLine("反時計回り認識");
        mPrevGestureTime = counterClockwiseTime;
    }
}
```

3.4. オーディオプレイヤー

3.3 で，ジェスチャーを認識させてコンソールに結果を表示できた．原理的には，認識後に実行される「処理 1」をオーディオ操作命令とすればオーディオプレイヤーができる．ただし，実際にはコンソールアプリケーションを GUI を持ったアプリケーションに変更するにはプログラムの大きな変更が必要になる．そのため，以下ではその変更が比較的小さい C#のみに限定し，概略の説明を行う．

C#で作製したオーディオプレイヤーのレイアウトを図 3.2 に示す．

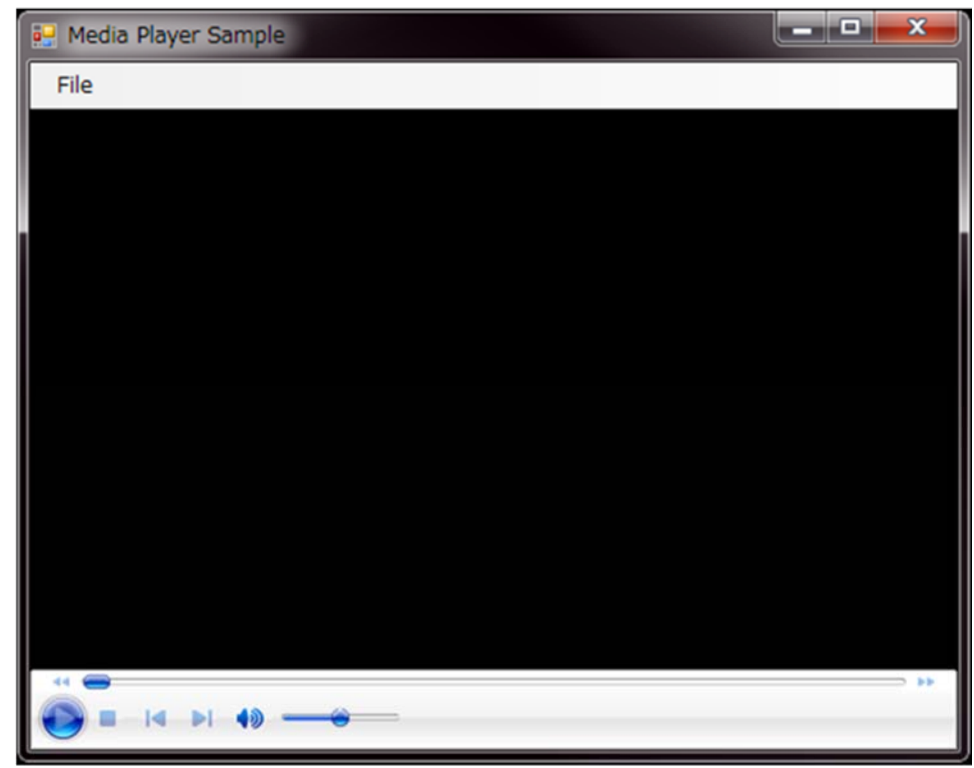

図 3.2　オーディオプレイヤーレイアウト

操作項目に対するジェスチャーは表 3.1 に示したように，上スワイプで曲の再生及び一時停止，下スワイプで曲の停止，左スワイプで前の曲に戻し，右スワイプで次の曲に移る．音量の調節は時計回りのサークルで音量をあげ，反時計回りのサークルで音量を下げることができるように設定する．

表 3.1　操作割り当て

操作項目	ジェスチャー
再生	上スワイプ
一時停止	上スワイプ
停止	下スワイプ
次の曲に進む	右スワイプ
前の局に戻る	左スワイプ
音量上げる	時計回りサークル
音量下げる	反時計回りサークル

まず，音楽操作に必要な関数を delegate 型で定義する．それぞれに担わせる役割は下記の通りである．

- PressPlayDelegate()：曲の再生，一時停止
- PressStopDelegate()：曲の停止
- PressBackDelegate()：前の曲に戻る
- PressNextDelegate()：次の曲に進む
- VolumeUpDelegate()：音量を上げる
- VolumeDownDelegate()：音量を下げる

```
delegate void PressPlayDelegate();
delegate void PressStopDelegate();
delegate void PressBackDelegate();
delegate void PressNextDelegate();
delegate void VolumeUpDelegate();
delegate void VolumeDownDelegate();
```

さらに，それぞれの操作に対する命令を設定する．

PressPlay()では，playStateで音楽が再生中かそうでないかを取得し，再生中であればCtlcontrols.pause()関数で音楽を一時停止させる．音楽を再生していない場合にはCtlcontrols.play()で音楽を再生する．PressStop()では，Ctlcontrols.stop()で音楽の停止をする．PressBack()では，Ctlcontrols.previous()で前の曲に戻す．PressNext()では，Ctlcontrols.next()で次の曲に進む．VolumeUpでサークルジェスチャーを1回認識すると音量が5ずつ上がるように設定する．このとき，100を超える場合には100にする．VolumeDownでも同様に音量を5ずつ下げる．このとき0未満になる場合には0に設定する．

```
void PressPlay(){
   if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPlaying){
axWindowsMediaPlayer1.Ctlcontrols.pause();
   }else if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsStopped
          || axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPaused){
      axWindowsMediaPlayer1.Ctlcontrols.play();
   }
}
void PressStop(){
   axWindowsMediaPlayer1.Ctlcontrols.stop();
}
void PressBack(){
   axWindowsMediaPlayer1.Ctlcontrols.previous();
}
void PressNext(){
   axWindowsMediaPlayer1.Ctlcontrols.next();
}
void VolumeUp(){
   int volume = axWindowsMediaPlayer1.settings.volume;
      volume = volume + 5;
   if (volume > 100){
      volume = 100;
   }
   axWindowsMediaPlayer1.settings.volume = volume;
}
void VolumeDown(){
   int volume = axWindowsMediaPlayer1.settings.volume;
      volume = volume - 5;
   if (volume < 0){
      volume = 0;
   }
   axWindowsMediaPlayer1.settings.volume = volume;
}
```

これらのプレイヤー操作で 3.3.1 や 3.3.4 で記述した「処理 1」に相当する箇所に表 3.1 の通りに割り当てる．

```
if (leftSwipeStopped) { // 左スワイプが認識されたら
    if (leftSwipeTime - mPrevLeftSwipeTime > detectionWaitTime) { // 以前の左スワイプ認識
より1秒以上たっていたら
        form.Invoke(new PressBackDelegate(form.PressBack)); // 前の曲に戻る
        mPrevLeftSwipeTime = leftSwipeTime;
    }
}
if (rightSwipeStopped) { // 右スワイプが認識されたら
    if (rightSwipeTime - mPrevRightSwipeTime > detectionWaitTime) { // 以前の右スワイプ認
識より1秒以上たっていたら
        form.Invoke(new PressNextDelegate(form.PressNext)); // 次の曲へ
        mPrevRightSwipeTime = rightSwipeTime;
    }
}
if (upSwipeStopped) { // 上スワイプが認識されたら
    if (upSwipeTime - mPrevUpSwipeTime > detectionWaitTime) { // 以前の上スワイプ認識より
1秒以上たっていたら
        form.Invoke(new PressPlayDelegate(form.PressPlay)); // 曲の再生、一時停止
        mPrevUpSwipeTime = upSwipeTime;
    }
}
if (downSwipeStopped) { // 下スワイプが認識されたら
    if (downSwipeTime - mPrevDownSwipeTime > detectionWaitTime) { // 以前の下スワイプ認識
より1秒以上たっていたら
        form.Invoke(new PressStopDelegate(form.PressStop)); // 曲の停止
        mPrevDownSwipeTime = downSwipeTime;
    }
}
if (clockwiseStopped) { // 時計回りがが認識されたら
    if (clockwiseTime - mPrevClockwiseTime > detectionWaitTime) { // 以前の時計回り認識よ
り1秒以上たっていたら
        form.Invoke(new VolumeUpDelegate(form.VolumeUp)); // 音量を上げる
        mPrevClockwiseTime = clockwiseTime;
    }
}
if (counterClockwiseStopped) { // 時計回りがが認識されたら
    if (counterClockwiseTime - mPrevCounterClockwiseTime > detectionWaitTime) { // 以前の
反時計回り認識より1秒以上たっていたら
        form.Invoke(new VolumeDownDelegate(form.VolumeDown)); // 音量を下げる
        mPrevCounterClockwiseTime = counterClockwiseTime;
    }
}
```

以上でオーディオプレイヤーのジェスチャーによる操作が可能となる．

3.5. プログラムの改良

3.4 で述べたオーディオプレイヤーのプログラムでは，Leap Motion がジェスチャーの停止を認識したときのみ，プレイヤー操作が可能であった．そのため，音量操作を行う際に円を何周描いても 1 度のみの認識となり，音量を大きく変更することに時間がかかる．

そこで，プログラムを改良し，ジェスチャー停止を認識しなくとも円を半周描くごとに音量調整が行われるように変更した．

このとき，円を何周描いているかを判別する progress 関数を利用し円の描画状況を把握する．

プログラムの変更点を以下に示す．

まず，ジェスチャーの状態を表す変数の名称を clockwaiseStopped，counterClockwiseStopped から clockwiseUpdating, counterClockwiseUpdating に変更する．さらに現在の円の半周を何回描いたかを格納する整数変数 Progress を宣言する．なお，Progress の以前の値を格納する変数 Progress0 も同様に宣言されている．

```
bool clockwiseUpdating = false;
bool counterClockwiseUpdating = false;

int Progress; //現在のプログレス
```

次に，ジェスチャーの状態の判定を，ジェスチャーの停止である STATE_STOP からジェスチャー継続中である STATE_UPDATE に変更する．現在のサークルジェスチャーの進行状況である progress を 0.5 で割り，これを int 型（整数型）にすることで半周毎に 1 ずつ増える変数となる（図 3.3）．

Progress0 と Progress に差が生じたタイミングでサークルジェスチャーを認識する．その後 Progress0 に Progress を格納し，サークルジェスチャーが停止するまでこの処理を繰り返すと半周毎に音量を増減することができる．

```
// 認識したCIRCLEジェスチャーの履歴を表示する
foreach (CircleGesture gesture in mCircleGestureList){
    if (gesture.State == Leap.Gesture.GestureState.STATE_UPDATE){//サークル継続認識
          Progress = (int)(gesture.Progress / 0.5);

    if (gesture.Pointable.Direction.AngleTo(gesture.Normal) <= Math.PI / 2){
       if (Progress != Progress0){
          clockwiseUpdating = true;
          }
        }
        else{
       if (Progress != Progress0){
          counterClockwiseUpdating = true;
          }
        }
        Progress0 = Progress;
    }
}
```

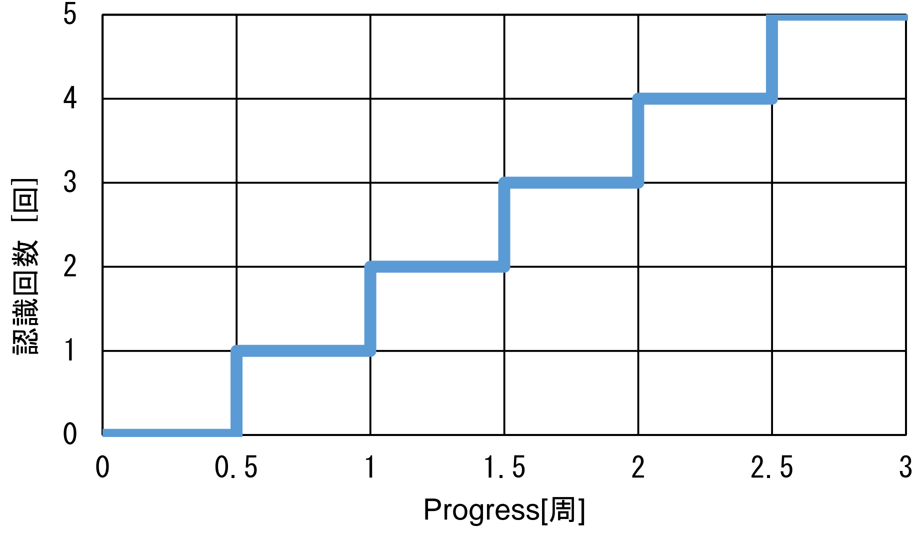

図 3.3 Progress と認識回数の関係

第4章　　認識率実験

本章では，自動車内での Leap Motion を設置する位置を決めるための実験を行う．使用するプログラムは前章で用いたコンソール画面に認識されたジェスチャーを表示するプログラムである．想定した設置場所は，シフトレバー上の車内 A，ハンドルとメーターの間の車内 B である（図 4.1）．

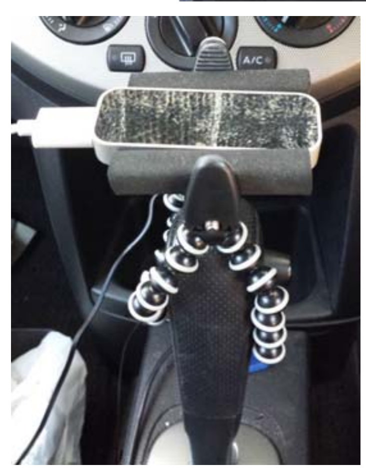

図 4.1　想定した設置位置
（上：2 つの設置位置，左下：車内 A，右下：車内 B）

4.1. 実験方法

上下左右方向のスワイプジェスチャーと時計回り，反時計回りのサークルジェスチャーの計 6 種類のジェスチャーを 1 セットとして 20 セット 120 回ジェスチャーを行う.

Leap Motion を研究室の卓上，車内 A，車内 B に設置し，それぞれの位置でジェスチャーを行う. 卓上及び車内 B では両手で, 車内 A では左手のみでジェスチャーを行う.

このとき対象人数は開発者である私とそれ以外の 6 名で，各設置場所で 20 セット行った.

また，卓上ではデスクトップ PC を使用し，車内ではノート PC を持ち込んで実験を行った．それぞれの PC の性能を表 4.1 に示す.

表 4.1　使用した PC の性能比較

	デスクトップ PC（卓上）	ノート PC（自動車内）
CPU	Intel®Core™i7-3770 3.40GHz	Intel®Core™i5-2467M 1.6GHz
メモリ	8.00GB	4.00GB

4.2.　結果及び考察

本論文では，狙ったジェスチャーの認識を認識成功，狙ったジェスチャー以外のジェスチャーの認識を誤認識とし，ジェスチャーが認識されなかったものを認識無しとする．

4.2.1.　認識率

開発者の結果（青），及び被験者 6 名の平均（オレンジ）をサークルジェスチャーについてまとめたのが図 4.2 である．開発者の結果を見ると卓上左手では 100%，卓上右手では 90%認識できた．車内 A では卓上左手と比較すると認識率は 15%減の 85%だった．

また，被験者の卓上では左手で 72.1%，右手では 72.5%認識された．車内 A でも 73.3%認識でき，卓上との差が見られなかった．

開発者の認識率が高いことから，このジェスチャー操作には慣れが必要であるとことがわかる．

車内 B では開発者，被験者共にその他の位置と比べ認識率が著しく低くなった．

この原因として車内 B ではハンドルやメーターパネルの庇部分などの障害物があり，ジェスチャー認識以前に，手を検出することが困難で，手が検出されていない状態でジェスチャーをしていたためと考えられる．

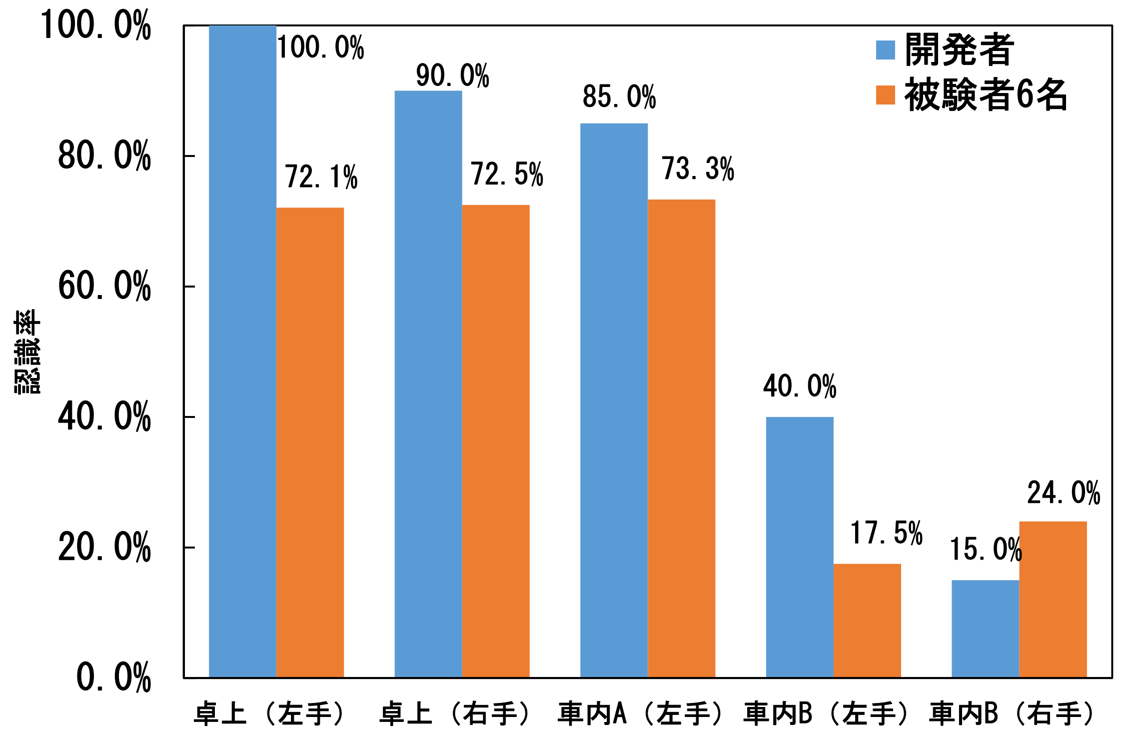

図 4.2　認識率実験結果（サークルジェスチャー）

開発者の結果（青），及び被験者 6 名の平均（オレンジ）をスワイプジェスチャーについてまとめたのが図 4.3 である．開発者の結果を見ると卓上左手では 90%，卓上右手では 83.8%認識された．車内 A では卓上左手と比較すると 18.7%減の 71.3%だった．

また，被験者の卓上では左手で 67.3%，右手では 63.3%認識された．車内 A では 48.8%で卓上と比較し 18.5%低下した．開発者と被験者に共通して言えるのはサークルジェスチャーよりもスワイプジェスチャーの方が難易度が高いことである．

車内 B では開発者，被験者共にサークルジェスチャーと同様にその他の位置と比べ認識率が著しく低くなった．

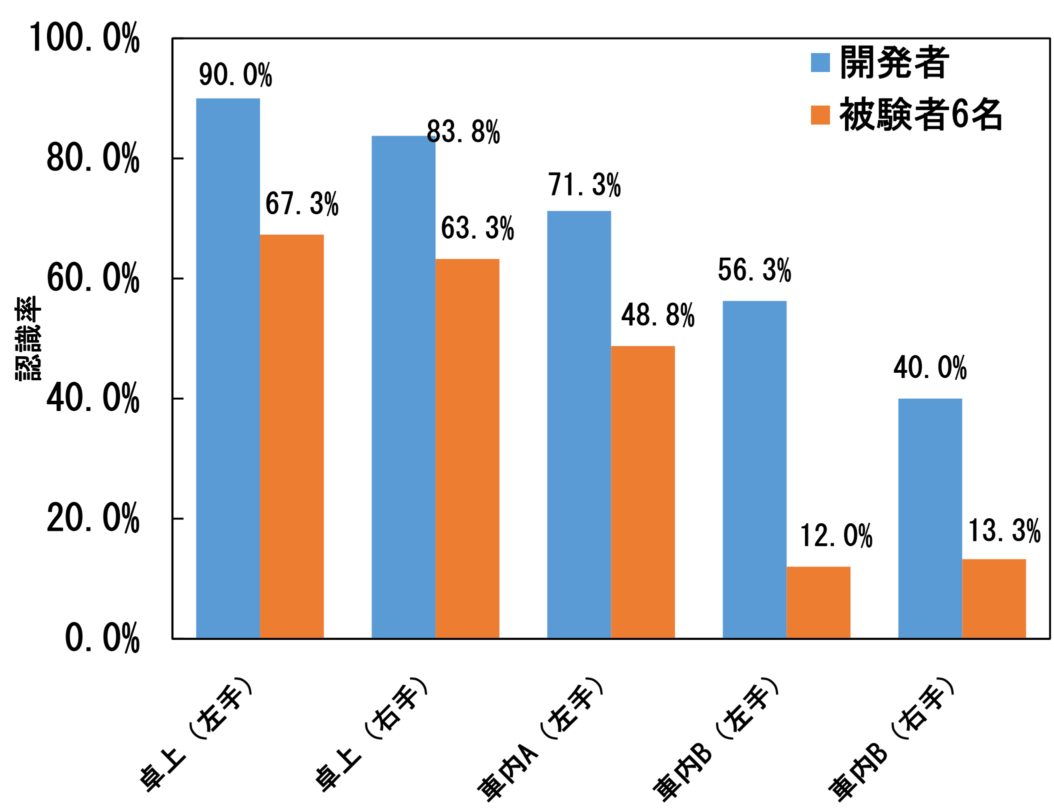

図 4.3　認識率実験結果（スワイプジェスチャー）

以上の結果をもとに，自動車内の Leap Motion の設置位置は車内 A に設置することに決定した．

また，設置環境，ジェスチャー動作を行う環境の両面で最も適していたと考えられる卓上での実験でも被験者の結果では約 70%の認識率しか得られなかった．そこで，認識パラメータを調整し認識率を上げる必要があると考えた．

また，車内での認識率を卓上での結果に近付ける必要があり，認識パラメータの調整に加え，設置環境を改善する必要があると考えた．

これらの改良については第 5 章で述べる．

4.2.2. 誤認識

それぞれのジェスチャーにおいて，狙ったジェスチャーが認識されない誤認識により，オーディオ操作時に誤動作が起こり得る．本節では誤認識の詳細と，その原因について触れる．

- サークルジェスチャー

図 4.4 に開発者と被験者 6 名のサークルジェスチャーの誤認識の内訳を示す．
データは卓上左手の結果のみを用いた．
被験者では「スワイプの誤認識」が 20%あり，「認識無し」が 7.9%となっている．

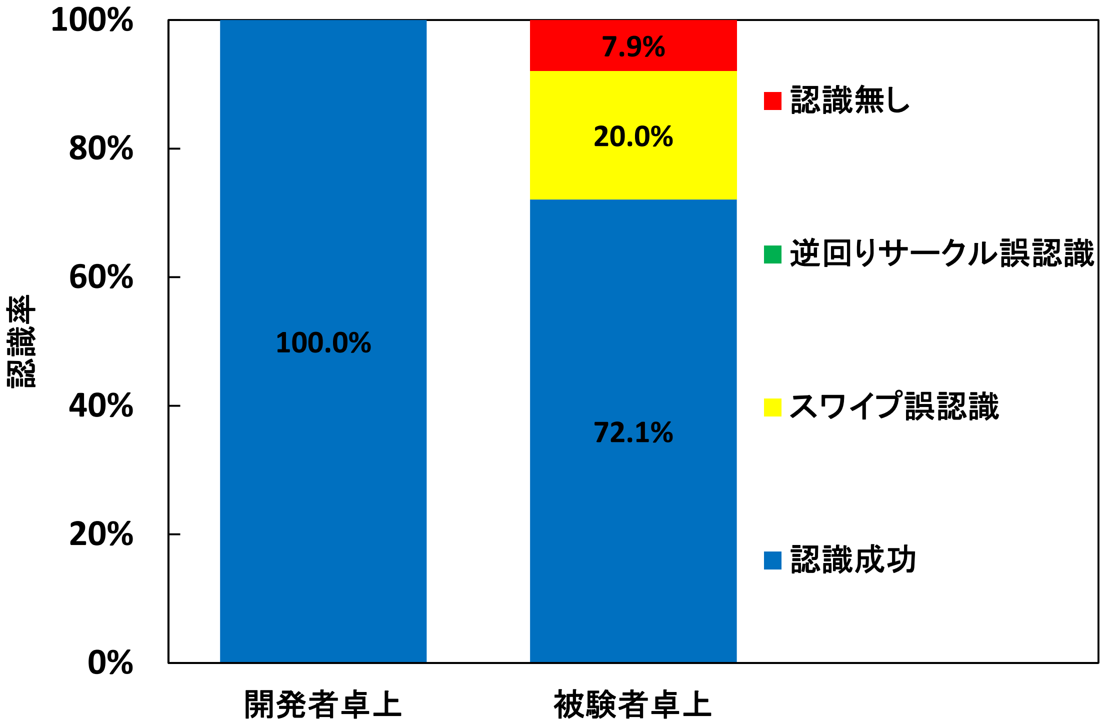

図 4.4　開発者，被験者のサークルジェスチャーの誤認識の内訳

● スワイプジェスチャー

図 4.5 に開発者と被験者 6 名のスワイプジェスチャーの誤認識の内訳を示す．

先ほどと同様，データは卓上左手のみを用いた．

開発者では「認識無し」が 7.5%，被験者では 20.0%とどちらも最大の失敗原因となり，スワイプジェスチャーはサークルジェスチャーに比べて認識が難しいことがわかる．また，「サークルの誤認識」が開発者では 2.5%あり，被験者では 6.3%だった．狙ったスワイプの「逆向きスワイプの誤認識」が被験者では 5.0%あった．

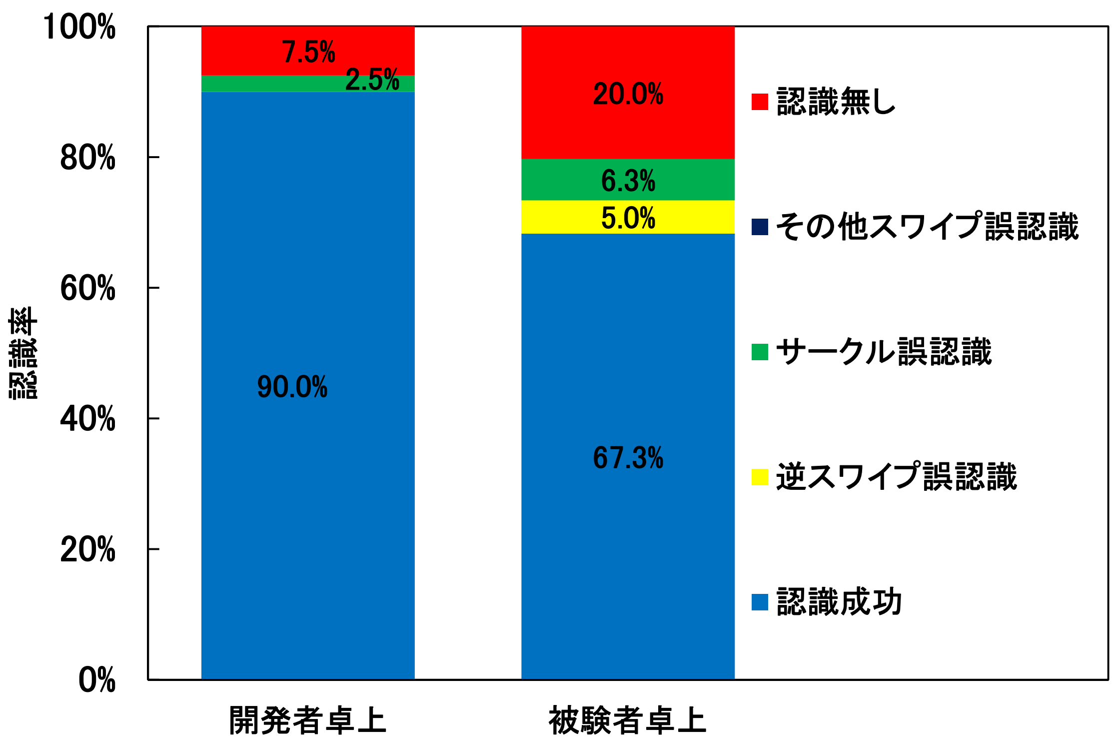

図 4.5 開発者，被験者のスワイプジェスチャーの誤認識の内訳

以上のようにジェスチャーの失敗や誤認識の中で多かったものには，

- ジェスチャーの認識無し
- スワイプジェスチャー時に逆向きのスワイプを誤認識
- スワイプジェスチャー時にサークルジェスチャーを誤認識
- サークルジェスチャー時にスワイプジェスチャーを誤認識
- ジェスチャーをしていない時にサークルを誤認識

の 5 点がある．

これらの誤認識は以下の原因で起こると考えられる．

・　ジェスチャーの認識無し

適切な位置でジェスチャーをしていないことや，ジェスチャーとして認識される条件を満たしていないことが原因で，ジェスチャーが認識されない．

・　スワイプジェスチャー時にサークルジェスチャーを誤認識

狙っていたスワイプが認識されず，指を元の位置に戻すとき小さな円を描いてしまい，それをサークルとして誤認識してしまう（図 4.6）．

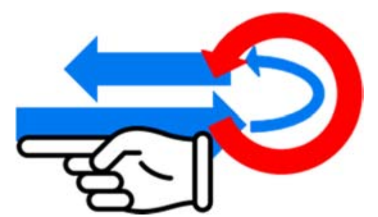

図 4.6　スワイプ時にサークル誤認識

・　スワイプジェスチャー時に逆向きのスワイプを誤認識

狙っていたスワイプが認識されず，指を元の位置に戻すときスワイプを誤認識してしまう（図 4.7）．

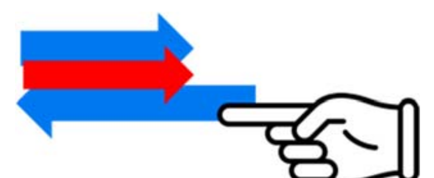

図 4.7　スワイプ時に逆向きスワイプの誤認識

・　サークルジェスチャー時にスワイプジェスチャーを誤認識

円を描く際水平方向や垂直方向に移動したとき，スワイプを誤認識してしまう（図 4.8）.

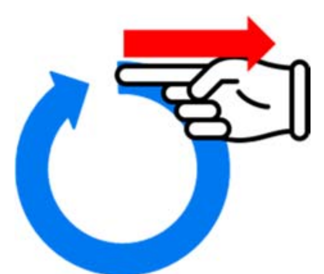

図 4.8　サークル時にスワイプを誤認識

・　ジェスチャーをしていない時にサークルを認識

ジェスチャー待機時に指の微細な動きで小さな円を描いてしまい，それをサークルジェスチャーとして誤認識してしまう（図 4.9）.

図 4.9　ジェスチャーしてないときサークルを誤認識

これら誤認識の減少と認識率の向上を目指し，認識パラメータの調整が必要である.

第5章　認識率の向上

5.1. 認識パラメータ調整

4 章で見たように，ジェスチャーによる車載機器の操作を実現するため，下記のように認識パラメータを調整し，認識率を向上させることを試みた.

- スワイプジェスチャー時の逆向きスワイプの誤認識
  狙ったスワイプを認識させるため，スワイプを認識しやすくなるように，認識するスワイプの最小の長さを 150[mm]から 100[mm]に短くする.

- サークルジェスチャー時のスワイプジェスチャーの誤認識
  スワイプを認識しにくくするために，認識する最小の速度を 1000[mm/s]から 1300[mm/s]に大きくする.

- スワイプジェスチャー時のサークルジェスチャーの誤認識
  サークルを認識しにくくするために，認識する最小の半径を 5[mm]から 30[mm]に大きく，最小の角度を 1.5π[rad]から 1.7π[rad]に大きくする.

それぞれのパラメータの変更点を表 5.1，表 5.2 に示す.

表 5.1　スワイプジェスチャー認識条件変更

	初期値	変更後
最小長さ	150[mm]	100[mm]
最小速度	1000[mm/s]	1300[mm/s]

表 5.2　サークルジェスチャー認識条件変更

	初期値	変更後
最小半径	5.0[mm]	30[mm]
最小角度	1.5π[rad]	1.7π[rad]

これらのパラメータは，setFloat()により設定できる．

- Gesture.Swipe.MinLength　：スワイプを認識する最小の長さを変更
- Gesture.Swipe.MinVelocity　：スワイプを認識する最小の速度を変更
- Gesture.Circle.MinRadius　：サークルを認識する最小の半径を変更
- Gesture.Circle.MinArc　：サークルを認識する最小の角度を変更

このとき，最後に leap.config().save()を呼び出すことで設定を反映させる．

C++
```
//設定を変える
leap.config().setFloat("Gesture.Swipe.MinLength", 100.0);
leap.config().setFloat("Gesture.Swipe.MinVelocity", 1300.0);
leap.config().setFloat("Gesture.Circle.MinRadius", 30.0);
leap.config().setFloat("Gesture.Circle.MinArc", 1.7*Leap::PI);
leap.config().save();
```

5.2. パラメータ調整後認識率

パラメータ調整後のプログラムでの認識率を調べ，調整前の認識率と比較する.

5.2.1. 実験方法

4.1 と同様のジェスチャーを，卓上左手と車内 A の位置で行い，結果を記録する.
このとき対象は開発者と被験者 6 名で，各設置場所で 20 セット行った.

5.2.2. 結果及び考察

5.2.2.1. 認識率

パラメータ調整前後の，卓上における開発者のサークルジェスチャーの結果を図 5.1 に示す. パラメータを調整する前後の認識率を比較すると，調整前はすべて成功していたが，調整後では「認識無し」がサークルジェスチャー40 回中 1 回の 2.5%,「スワイプの誤認識」が 3 回の 7.5%発生してしまった.

5.1 で示したように，サークルジェスチャーが認識しにくくなるようパラメータを調整したことが原因にあると考えられる．また，100%と言う結果から見て試行回数が少なかったことも考えられる.

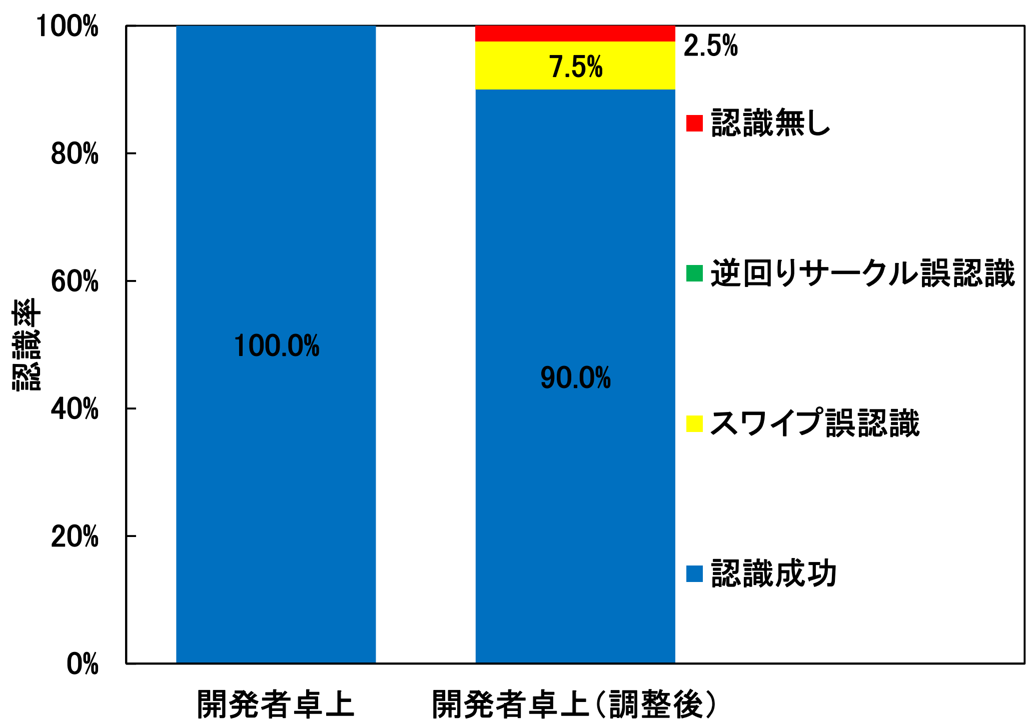

図 5.1 パラメータ調整前後のサークルジェスチャーの比較（開発者）

パラメータ調整前後の，卓上における被験者 6 名のサークルジェスチャーの結果を図 5.2 に示す．サークル時に「スワイプの誤認識」が 20%から 13.3%に減少したものの，「認識無し」が増加し，サークルジェスチャーの認識率に変化は見られなかった．しかし，オーディオの誤動作の原因となるジェスチャーの誤認識は減少したと言える．

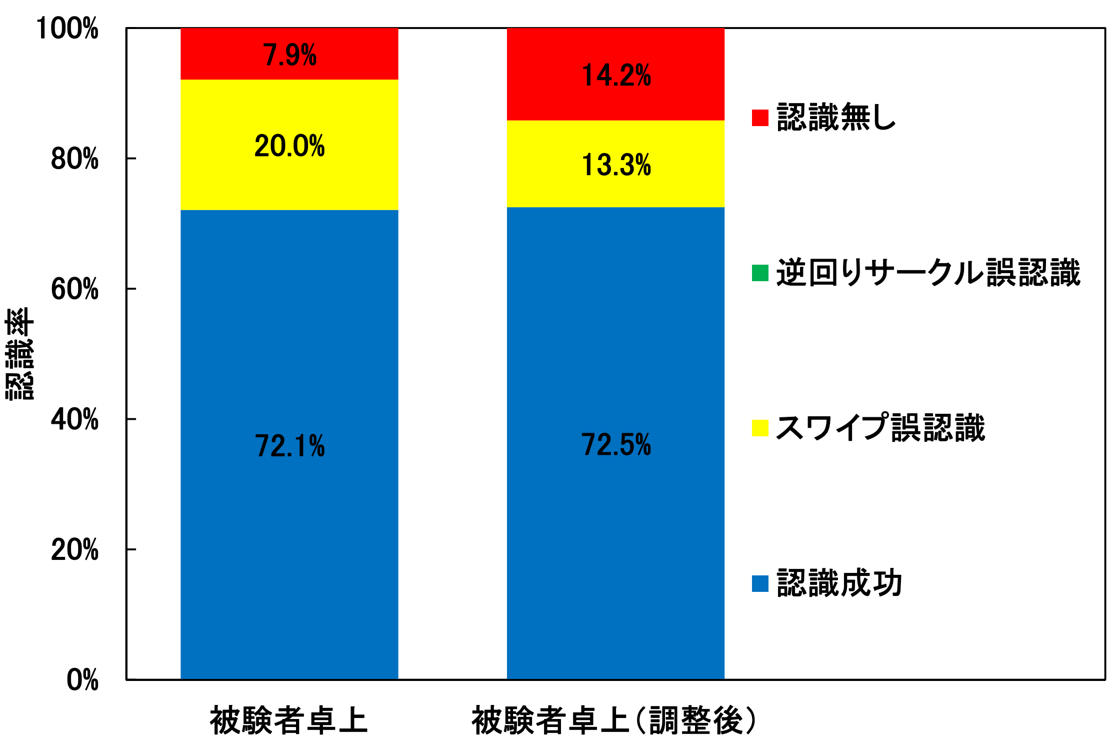

図 5.2　パラメータ調整前後のサークルジェスチャーの比較（被験者 6 名）

パラメータ調整前後の，卓上における開発者のスワイプジェスチャーの結果を図 5.3 に示す．調整前にあった「認識無し」，「逆向きサークルの誤認識」が無くなった．

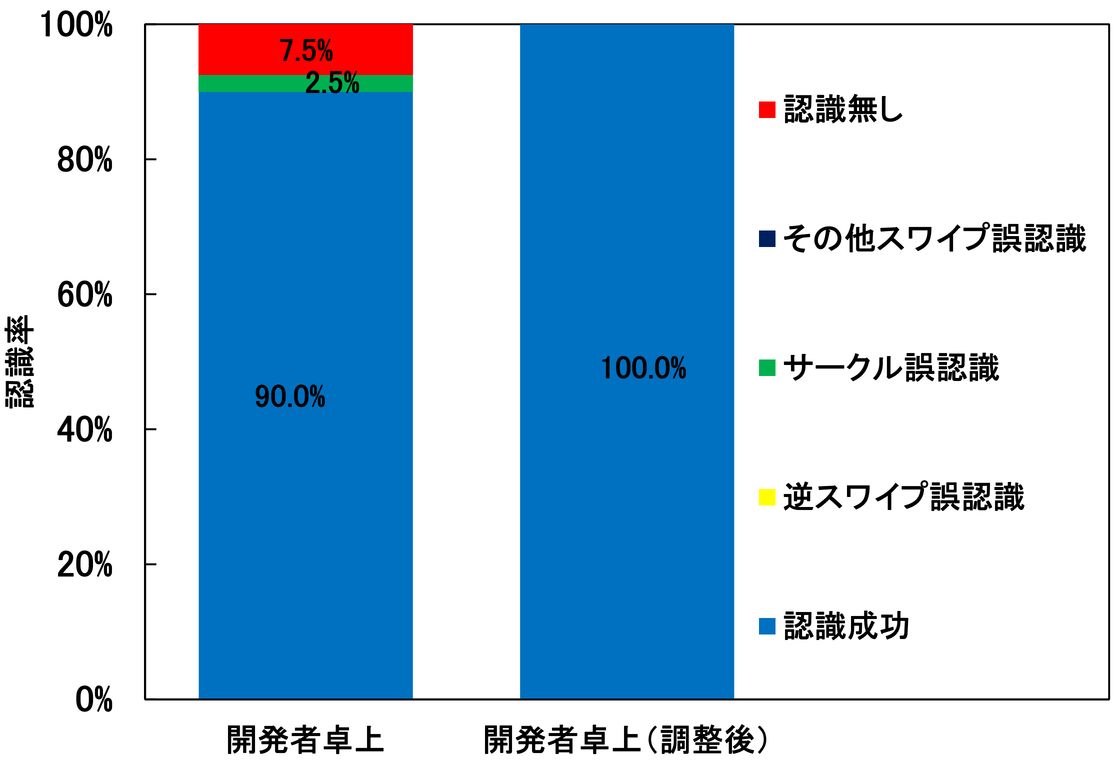

図 5.3　パラメータ調整前後のスワイプジェスチャーの比較（開発者）

パラメータ調整前後の，卓上における被験者 6 名のスワイプジェスチャーの結果を図 5.4 に示す．調整前後で比較するとスワイプ時の「サークルの誤認識」が無くなり，「逆スワイプの誤認識」には変化が無かったが，「認識無し」が減り，スワイプの認識率が向上した．

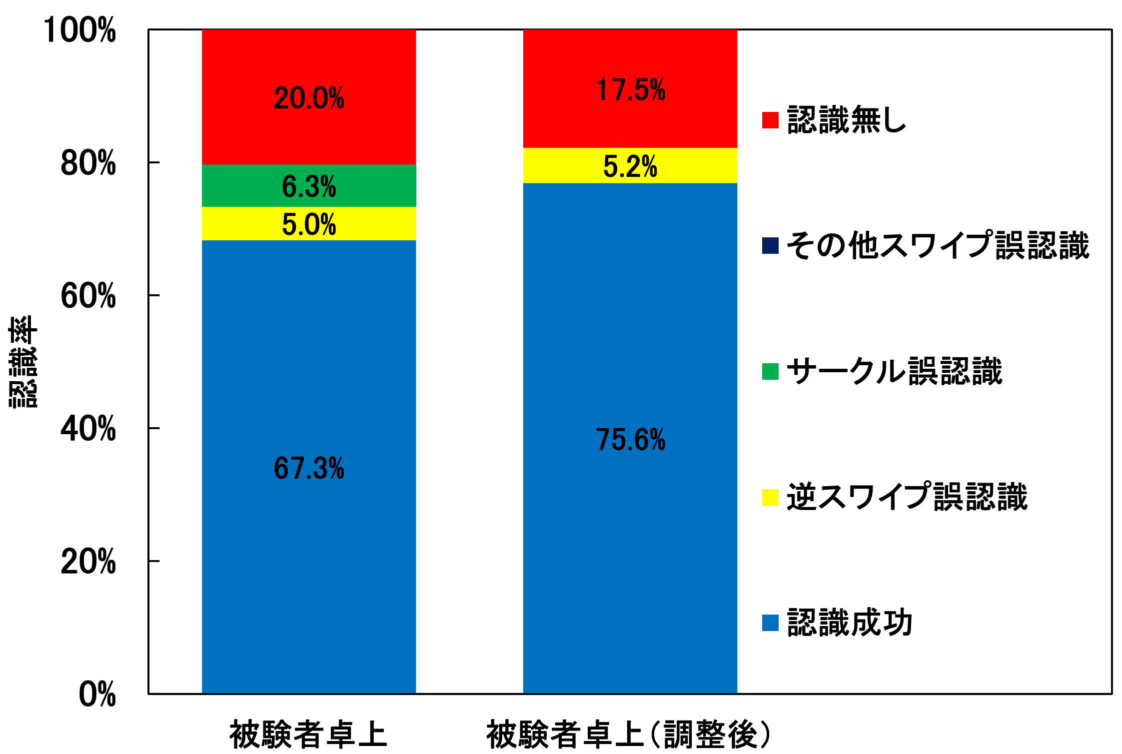

図 5.4　パラメータ調整前後のスワイプジェスチャーの比較（被験者 6 名）

まとめると，サークルジェスチャーでは認識の難易度をあげたため，サークルジェスチャーの認識率に向上は見られなかった．しかしスワイプジェスチャーにおいて「サークルの誤認識」は無くなり，認識率も向上した．

認識パラメータをさらに調整し，最も適したパラメータを見つけることができれば，より認識率が向上すると考えられる．

5.3. 認識率に影響する要因

図 4.2 と図 4.3 を比較すると車内 A の結果は卓上のものに比べ，認識率が低い傾向があることがわかる．

パラメータの調整をし，認識率の実験を行っていく中で，設置環境などプログラム以外に，以下 2 点の要素が認識率に影響すると推測した．

- 設置角度による差
- PC の性能による差

これらの原因による認識率の変化について以下で調べる．

5.3.1. 設置角度による認識率の比較

運転席に座り、車内 A に設置した Leap Motion 上に手をかざすと図 5.5 のように左手の向きは身体の向きに対して左側に 10 度ほどずれるため，自動車内に設置する際，Leap Motion を左側に角度をつけて設置した方が認識率の向上が狙えるのではないかと推測した．

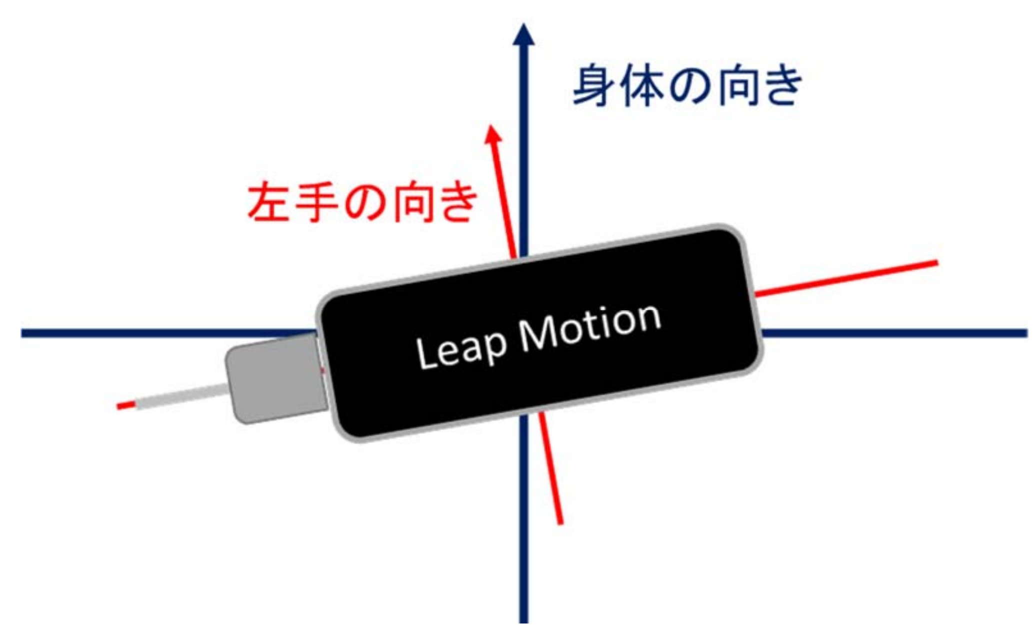

図 5.5　身体の向きに対する左手の向き

そこで自動車内を想定し，研究室内で身体の左側，シフトレバー付近になるよう Leap Motion を設置し，角度を身体の向きに対し右側 10 度から左側 30 度まで 10 度ずつ角度を変え認識率を調べた．

このとき，慣れによる影響を無くすため対象は開発者のみで，各ジェスチャーを 20 セット行った．

このときのサークルジェスチャーとスワイプジェスチャーの結果を全て合算した認識率が図 5.6 である．結果を見ると身体の向きに沿って設置したときが最も認識率が良く，左側に角度を付けるごとに認識率が低くなっている．また，右側に角度をつけると認識率が低くなることもわかった．この結果より，先に述べた角度をつけて Leap Motion を設置した方が，認識率が良くなるという推測は間違っており，実際は身体の向きに沿って設置するのが最も適していると言える．

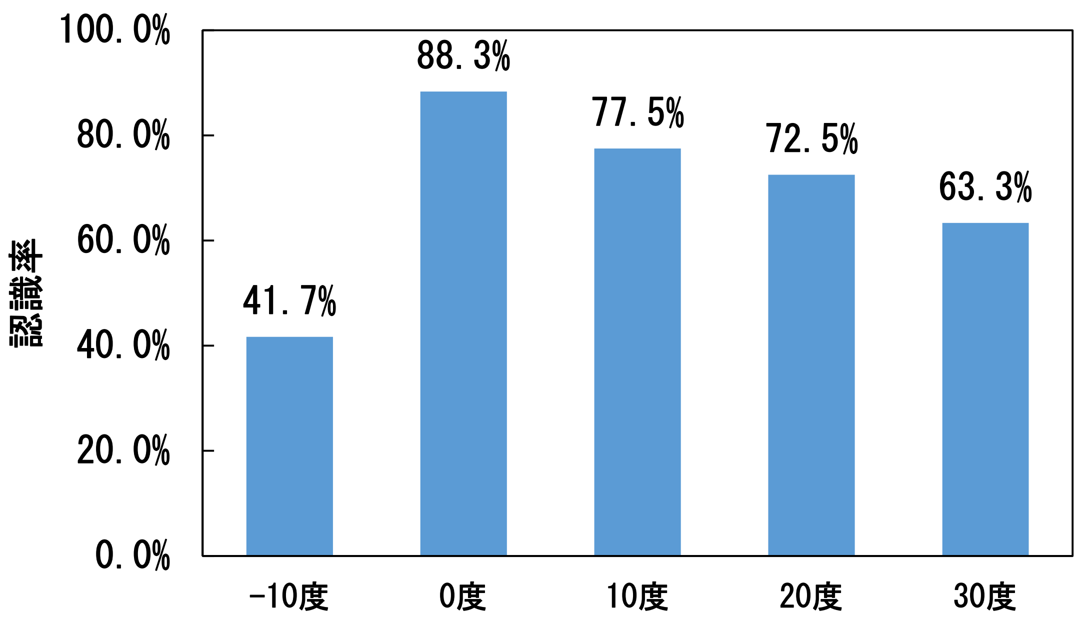

図 5.6　設置角度による認識率の変化

5.3.2. PC の違いによる認識率の比較

これまでの実験では，卓上での実験ではデスクトップ PC を使用し，車内での実験ではノート PC を使用した．

PC の違いによる認識率の差の有無を調べるため，卓上にてデスクトップ PC，ノート PC それぞれに接続した Leap Motion 2 台を図 5.7 のように並べ，その中央付近で同時にジェスチャーを認識させ，開発者のみで 20 セットジェスチャーをして認識率の違いを調べた．

図 5.7 Leap Motion を並べた様子

図 5.8 に結果を示す．同時にジェスチャーを認識させたにもかかわらず，デスクトップ PC の認識率は平均 95.4%，ノート PC の認識率は 83.8%と結果は大きく異なった．

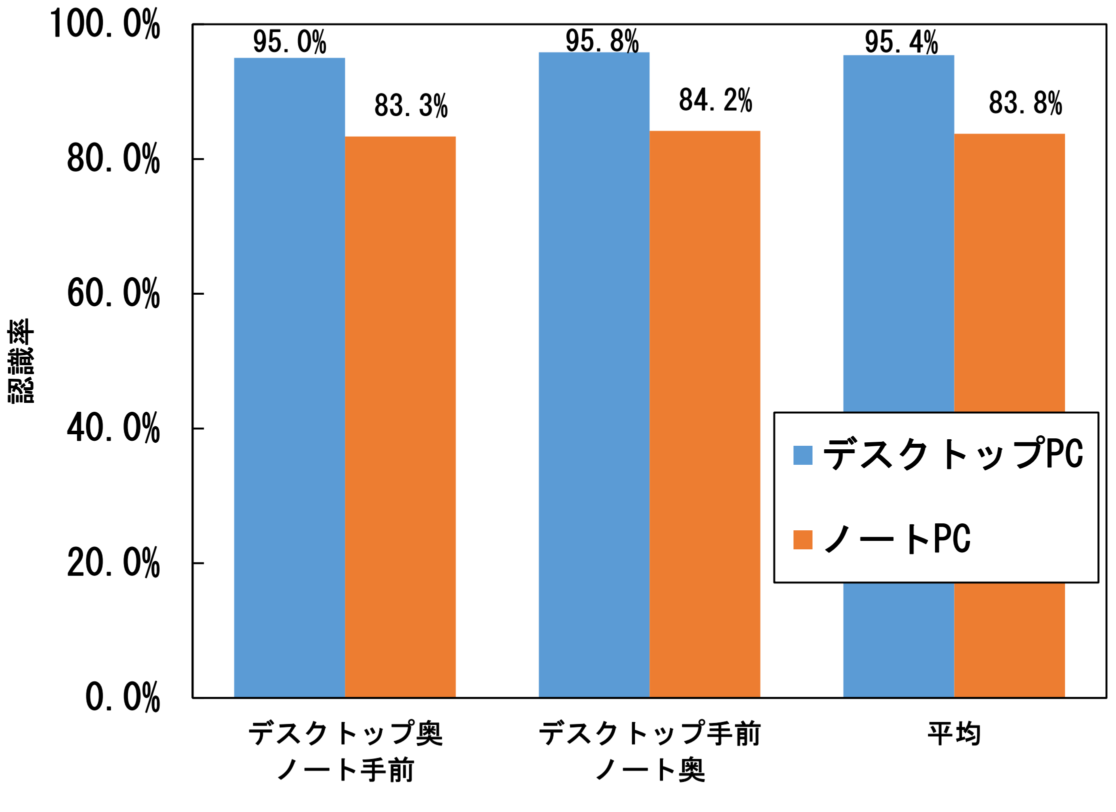

図 5.8　PC の違いによる認識率

この原因は PC の処理能力の差であると推測し，それぞれの PC での 1 秒間のフレーム更新回数を表示させるプログラムを作製し，比較した．

その詳細については 5.3.3 で述べる．

5.3.3. フレーム数の表示

PC の性能の違いによる認識率の変化を知るため，Leap Motion によるデータ取得のフレーム数を計測する．Leap Motion では現在のフレームと合わせ過去 60 フレーム，つまり 59 フレーム前までを記録することができる．これを利用し，過去の指定したフレームと現在のフレームとの間の時間から，フレームレートを計測した．

以下にプログラムの変更点を示す．

以前のフレームとして PreviousFrame を宣言する．また以前のフレームの時間とし mPrevFrameTime を宣言する．

C++
```
Leap::Frame PreviousFrame;

int64_t mPrevFrameTime; //前のフレーム取得時間
```

onFrame 関数内の

フレームの更新をさせている部分で leap.frame()の（）内に入れた値分前のフレームを記録する．

C++
```
// フレームの更新
mLastFrame = mCurrentFrame;
mCurrentFrame = leap.frame(); //現在のフレーム
PreviousFrame = leap.frame(59); //59フレーム前のフレーム
```

現在のフレームの時間を timestamp()で記録しその時間から 59 フレーム前の時間を引いた時間を表示させる．

C++
```
std::cout << mCurrentFrame.timestamp() - PreviousFrame.timestamp() <<std::endl;
```

このプログラムを使用してデスクトップ PC とノート PC の 1 秒間のフレーム更新回数を調べた．

その結果，デスクトップでは平均 113.64 フレーム，ノート PC では平均 108.56 フレームと約 5 フレーム分の差があった．これより図 5.8 の認識率の差は使用していた PC の処理能力の差によるものと考えられる．

第6章　結論

本研究では，車載機器を操作することが原因の脇見運転による事故の減少を目指し，オーディオプレイヤーをジェスチャーにより操作できるインターフェイスの作製を試みた．

2章ではオーディオプレイヤーの操作のために最低限必要な操作項目を挙げ，少なくとも 6 つのジェスチャーが必要であることがわかった．そこで，手の動きを認識できるセンサである Leap Motion で利用可能なジェスチャーの中から，認識が容易であるスワイプジェスチャー（4 方向）とサークルジェスチャー（時計回りと反時計回り）の計 6 種類を使用しオーディオプレイヤーの仕様を決定した．

3章ではジェスチャー認識用のコンソールアプリケーションと，オーディオ操作用の GUI アプリケーションを作製した．オーディオプレイヤーのアプリケーションの音量調節のジェスチャーが認識されるのは，当初サークルジェスチャーの停止のタイミングであった．しかし，これでは指で一周分の円を描くごとに指を停止しなければ音量を調整できず，音量の調整量が大きい場合に効率が悪い．そこで，サークルジェスチャーを連続的に認識できるよう改善したところ，音量の効率的な変更が可能になった．

4章では自動車内での Leap Motion の設置位置を決定するために認識実験を行った．その結果，現実のオーディオの操作パネル付近に設置したときの認識率が他の位置よりも高かったため，設置場所はオーディオの操作パネル付近に決定した．また，この実験において開発者である私の認識率が被験者の認識率よりも大きかったことから，ジェスチャー操作への習熟度も認識率に影響することがわかった．さらに，この認識率は，ジェスチャーに適した環境である研究室の卓上でさえ約 70%と小さく，認識率の向上が必要であることがわかった．

そこで，認識率向上のための準備として，各ジェスチャーの誤認識の原因を明らかにした．

5章では認識率の向上と誤認識の減少を目指し，ジェスチャーの認識パラメータの調整を行った．その結果，まずサークルジェスチャーでは認識率に変化は見られなかったものの，誤認識は減少した．また，スワイプジェスチャーでは認識率が 10%向上し，誤認識も減少した．

次に，認識率に影響すると考えられる Leap Motion の設置角度による認識率の変化を調べた．その結果，身体の向きと Leap Motion の向きに角度をつけずに設置したときが最も認識率が良くなることがわかった．

最後に，卓上で使用していたデスクトップ PC と車内で使用したノート PC の違いによる認識率の変化を調べた．デスクトップ PC での認識率が 95.4%，ノート PC の認識率が 83.8%と結果は大きく異なった．この原因は処理能力の差であると推測し，それぞれの PC での 1 秒間のフレーム更新回数を比較したところ，デスクトップ PC では秒間

113.64 フレーム，ノート PC では秒間 108.56 フレームで約 5 フレーム分の差があることがわかった．このフレーム更新回数の差が，認識率に反映されたのだと考えられる．

以上のように，ジェスチャーによってオーディオの操作が可能なインターフェイスを作製し，その認識率の向上を試みた．仕様に含めたオーディオ操作の範囲では一定の成功を収めたが，現状ではリストから曲を選ぶことができないなど，オーディオプレイヤーとしての操作が足りないという問題がある．これらの操作を追加するには，ジェスチャーの追加が必要になるが，それを行うと認識の難易度が上がり，認識率は下がると考えられる．その問題の解決は今後の課題である．

第7章　謝辞

研究活動及び本論文の作成にあたり，多くの助言や指導を賜りました金丸隆志准教授，本論文の執筆にあたり指導を賜りました疋田光孝教授，濱根洋人准教授に心より厚く御礼申し上げます．

研究テーマの決定に時間がかかり研究のスタートが送れ，プログラミングの知識が全く無い中，勉強用の教材や研究に関連する書籍を購入してもらい，プログラムの作成にあたって多くの助言やサポートをしていただき，重ねて感謝の意を表します．

また，同研究室の同輩，後輩．他研究室の同輩達にも実験の協力や相談に乗っていただき有意義な助言をしていただきました．

その他，私を支えてくださった皆様にも感謝の気持ちを表し，謝辞とさせていただきます．

参考文献

[1] 警視庁, 平成 26 年中の交通事故発生状況, 警視庁交通局.

[2] 東京海上日動 WIN クラブ, “わき見運転の危険性,” 2014.
http://www.tmn-win.com/win_plaza/open_pdf/w1409_safety.pdf.

[3] トヨタ自動車株式会社
http://toyota.jp/?ptopid=hea.

[4] 岸　則政, “車載情報システムにおける音声インターフェースの有効性と問題点,”
https://www.ipsj.or.jp/10jigyo/fit/fit2003/fit2003program/html/event/pdf/kishi.pdf.

[5] 武田　一哉, “自動車の中での音声認識,” 著: *IPSJ Magazine vol.45 No.10*.

[6] 任天堂株式会社
https://www.nintendo.co.jp/.

[7] Microsoft
https://www.microsoft.com/ja-jp/.

[8] 日本 HP
http://www8.hp.com/jp/ja/home.html.

[9] 中村　薫, Leap Motion プログラミングガイド.

[10] Leap Motion, “Leap Motion”
https://www.leapmotion.com/?lang=jp.

[11] Leap Motion, “Leap Motion SDK Documentation”
https://developer.leapmotion.com/.

図表目次

図 1.1　トヨタ社製音声認識マイク&スイッチ　[3]......6
図 1.2　左：センサーバーユニット，右：Wii リモコンと本体　[6]......7
図 1.3　Kinect　[7]......7
図 1.4　Leap Motion 本体......8
図 1.5　HP ENVY 17-j100 Leap Motion SE　[8]......8
図 2.1　Leap Motion 本体......11
図 2.2　Leap Motion の座標系　[11]......13
図 2.3　インタラクションボックス　[11]......13
図 2.4　スワイプジェスチャー　[11]......15
図 2.5　サークルジェスチャー　[11]......15
図 2.6　キータップジェスチャー　[11]......16
図 2.7　スクリーンタップジェスチャー　[11]......16
図 3.1　ジェスチャー認識のタイミング......21
図 3.2　オーディオプレイヤーレイアウト......28
図 3.3　Progress と認識回数の関係......33
図 4.1　想定した設置位置......34
図 4.2　認識率実験結果（サークルジェスチャー）......36
図 4.3　認識率実験結果（スワイプジェスチャー）......37
図 4.4　開発者，被験者のサークルジェスチャーの誤認識の内訳......39
図 4.5　開発者，被験者のスワイプジェスチャーの誤認識の内訳......40
図 4.6　スワイプ時にサークル誤認識......41
図 4.7　スワイプ時に逆向きスワイプの誤認識......41
図 4.8　サークル時にスワイプを誤認識......42
図 4.9　ジェスチャーしてないときサークルを誤認識......42
図 5.1　パラメータ調整前後のサークルジェスチャーの比較（開発者）......45
図 5.2　パラメータ調整前後のサークルジェスチャーの比較（被験者 6 名）......46
図 5.3　パラメータ調整前後のスワイプジェスチャーの比較（開発者）......47
図 5.4　パラメータ調整前後のスワイプジェスチャーの比較（被験者 6 名）......48
図 5.5　身体の向きに対する左手の向き......50
図 5.6　設置角度による認識率の変化......51
図 5.7　Leap Motion を並べた様子......52
図 5.8　PC の違いによる認識率......53

表 1.1　原付以上運転者の法令違反別交通事故件数（平成 26 年中）　[1]........5
表 2.1　Leap Motion の動作環境　[10]........12
表 2.2　Leap Motion と Kinect の比較　[7,10]........14
表 2.3　操作割り当て........17
表 3.1　操作割り当て........29
表 4.1　使用した PC の性能比較........35
表 5.1　スワイプジェスチャー認識条件変更........43
表 5.2　サークルジェスチャー認識条件変更........43

付録

1　　コンソールプログラム（C++）

```
#include "Leap.h"
#include <windows.h>
#include <list>

using namespace std;

stringstream ss;

class SampleListener : public Leap::Listener
{
public:

	bool mFirst;
	Leap::Frame mCurrentFrame;
	Leap::Frame mLastFrame;

	std::list<Leap::SwipeGesture> mSwipeGestureList;
	std::list<Leap::CircleGesture> mCircleGestureList;

	int64_t mPrevGestureTime; // 以前にジェスチャーを検出した時刻

	int mProgress0;

	// 変数初期化用コンストラクタ
	SampleListener(){
		mFirst = true;
		mPrevGestureTime = 0; // 以前にジェスチャーを検出した時刻
	}

	void onFrame(const Leap::Controller& leap)
	{

	//設定を変える(15.09.07)
	leap.config().setFloat("Gesture.Swipe.MinLength", 100.0);
	leap.config().setFloat("Gesture.Swipe.MinVelocity", 1300.0);
	leap.config().setFloat("Gesture.Circle.MinRadius", 30.0);
	leap.config().setFloat("Gesture.Circle.MinArc", 1.7*Leap::PI);
	leap.config().save();

	if(mFirst){
		mCurrentFrame = leap.frame();
		mFirst = false;
	}
	// フレームの更新
	mLastFrame = mCurrentFrame;
	mCurrentFrame = leap.frame();

	// 以前のフレームが有効であれば、今回のフレームまでの間に検出したジェスチャーの一覧を取得す
る
	// 以前のフレームが有効でなければ、今回のフレームで検出し他ジェスチャーの一覧を取得する
	auto gestures = mLastFrame.isValid() ? mCurrentFrame.gestures( mLastFrame ) :
mCurrentFrame.gestures();

	// 検出したジェスチャーの履歴を保存する
	for ( auto gesture : gestures ) {
		switch(gesture.type()){
			// SWIPEジェスチャーをmSwipeGestureListに登録
		case Leap::Gesture::TYPE_SWIPE:
```

```
                    {
        // IDによってジェスチャーを登録する
        auto it = std::find_if( mSwipeGestureList.begin(), mSwipeGestureList.end(),
        [gesture]( Leap::Gesture g ){ return g.id() == gesture.id(); } );
                if ( it != mSwipeGestureList.end() ) {
                *it = gesture;
                }
        else {     mSwipeGestureList.push_back( gesture );
                        }
                }
        break;

// CIRCLEジェスチャーをmCircleGestureListに登録
        case Leap::Gesture::TYPE_CIRCLE:
                {
// IDによってジェスチャーを登録する
auto it = std::find_if( mCircleGestureList.begin(), mCircleGestureList.end(),
        [gesture]( Leap::Gesture g ){ return g.id() == gesture.id(); } );
        if ( it != mCircleGestureList.end() ) {
                *it = gesture;
        }
        else {     mCircleGestureList.push_back( gesture );
                }
        }
        break;
        default:
        break;
                }
        }

        bool leftSwipeStopped = false;
        bool rightSwipeStopped = false;
        bool upSwipeStopped = false;
        bool downSwipeStopped = false;
        //bool clockwiseCircleStopped = false;
        //bool coclockwiseCircleStopped = false;
        bool clockwiseCircleUpdating = false;
        bool coclockwiseCircleUpdating = false;
        int64_t leftSwipeTime; // 左スワイプの検出時刻
        int64_t rightSwipeTime; // 右スワイプの検出時刻
        int64_t upSwipeTime; // 上スワイプの検出時刻
        int64_t downSwipeTime; // 下スワイプの検出時刻
        int64_t CircleTime; // 時計回りサークルを検出した時刻
        //int64_t coclockwiseCircleTime; // 反時計回りサークルを検出した時刻

        //これ以内の短い間隔のジェスチャーは検出しない。
        int64_t detectionWaitTime =  800000; // 1000*1000 (μ秒) = 1秒。

        float mProgress; //現在のプログレス

        // 検出したSWIPEジェスチャーの履歴を表示する
        for ( auto gesture : mSwipeGestureList ) {
                if( gesture.state()==Leap::Gesture::State::STATE_STOP){ // スワイプ停止の検出
                        if(gesture.direction().x<=-0.3){  // 左スワイプの検出
                                if(gesture.direction().y<=0.5){
                                        if(gesture.direction().y>=-0.5){
                                leftSwipeStopped = true;
                                leftSwipeTime = gesture.frame().timestamp();
                                        }
                                }
                        }else if(gesture.direction().x>=0.3){  //右スワイプの検出
                                if(gesture.direction().y<=0.5){
                                        if(gesture.direction().y>=-0.5){
```

```
							rightSwipeStopped = true;
							rightSwipeTime = gesture.frame().timestamp();
								}
							}
						}
						if(gesture.direction().y>=0.3){  // 上スワイプの検出
							if(gesture.direction().x<=0.5){
								if(gesture.direction().x>=-0.5){
							upSwipeStopped = true;
							upSwipeTime = gesture.frame().timestamp();
								}
							}
						}else if(gesture.direction().y<=-0.2){  //下スワイプの検出
							if(gesture.direction().x<=0.5){
								if(gesture.direction().x>=-0.5){
							downSwipeStopped = true;
							downSwipeTime = gesture.frame().timestamp();
								}
							}
						}
					}
		}


				// 検出したCICLEジェスチャーの履歴を表示する
				for ( auto gesture : mCircleGestureList ) {
						if( gesture.state()==Leap::Gesture::State::STATE_UPDATE){ // サークル継
続の検出
						mProgress = (int)((gesture.progress()/0.5)-0.5);
		if(gesture.pointable().direction().angleTo(gesture.normal()) <= (Leap::PI/2)){  // 時計回り
サークルの検出
							if(mProgress != mProgress0){
							clockwiseCircleUpdating = true;
							CircleTime = gesture.frame().timestamp();
							}
						}else {  //反時計回りサークルの検出
							if(mProgress != mProgress0){
							coclockwiseCircleUpdating = true;
							CircleTime = gesture.frame().timestamp();
								}
							}
						mProgress0 = mProgress;
						}
				}
		if(leftSwipeStopped){ // 左スワイプが検出されたら
				if(leftSwipeTime - mPrevGestureTime > detectionWaitTime){  // 以前の左スワイプ検出
より1秒以上たっていたら
				std::cout << "←  左スワイプ  ←" << std::endl;
				mPrevGestureTime = leftSwipeTime;
				}
		}
		if(rightSwipeStopped){ // 右スワイプが検出されたら
				if(rightSwipeTime - mPrevGestureTime > detectionWaitTime){  // 以前の右スワイプ検
出より1秒以上たっていたら
				std::cout << "→  右スワイプ  →" << std::endl;
				mPrevGestureTime = rightSwipeTime;
				}
		}
		if(upSwipeStopped){ // 上スワイプが検出されたら
				if(upSwipeTime - mPrevGestureTime > detectionWaitTime){  // 以前の上スワイプ検出よ
り1秒以上たっていたら
				std::cout << "↑  上スワイプ  ↑" << std::endl;
				mPrevGestureTime = upSwipeTime;
```

```
			}
		}
		if(downSwipeStopped){ // 下スワイプが検出されたら
			if(downSwipeTime - mPrevGestureTime > detectionWaitTime){  // 以前の下スワイプ検出
より1秒以上たっていたら
			std::cout << "↓  下スワイプ  ↓" << std::endl;
			mPrevGestureTime = downSwipeTime;
			}
		}

		if(clockwiseCircleUpdating){ // 時計回りサークルが検出されたら
			std::cout << "時計回り" << mProgress*0.5 << "周検出" << std::endl;
			mPrevGestureTime = CircleTime;
			}

		if(coclockwiseCircleUpdating){ // 反時計回りサークルが検出されたら
			std::cout << "反時計回り" << mProgress*0.5 << "周検出" << std::endl;
			mPrevGestureTime = CircleTime;
		}

		// 最後の更新から1秒たったジェスチャーを削除する(タイムスタンプはマイクロ秒単位)
		mSwipeGestureList.remove_if( [&]( Leap::Gesture g ){
			return (mCurrentFrame.timestamp() - g.frame().timestamp()) >= (1 * 1000 *
1000); } );
		// 最後の更新から1秒たったジェスチャーを削除する(タイムスタンプはマイクロ秒単位)
		mCircleGestureList.remove_if( [&]( Leap::Gesture g ){
			return (mCurrentFrame.timestamp() - g.frame().timestamp()) >= (1 * 1000 *
1000); } );
		}

		void onInit(const Leap::Controller& controller)
		{
			std::cout << __FUNCTION__ << std::endl;
		}

		void onConnect(const Leap::Controller&)
		{
			std::cout << __FUNCTION__ << std::endl;
		}

		void onDisconnect(const Leap::Controller&)
		{
			std::cout << __FUNCTION__ << std::endl;
		}

		void onExit(const Leap::Controller&)
		{
			std::cout << __FUNCTION__ << std::endl;
		}

		void onFocusGained(const Leap::Controller&)
		{
			//std::cout << __FUNCTION__ << std::endl;
		}

		void onFocusLost(const Leap::Controller&)
		{
			//std::cout << __FUNCTION__ << std::endl;
		}

		void onServiceConnect(const Leap::Controller&)
		{
			std::cout << __FUNCTION__ << std::endl;
```

```
	}

	void onServiceDisconnect(const Leap::Controller&)
	{
		std::cout << __FUNCTION__ << std::endl;
	}

	void onDeviceChange(const Leap::Controller& controller)
	{
		std::cout << __FUNCTION__ << std::endl;
	}
};

void main(){
	// リスナーを登録する
	// リスナーとのやり取りは別スレッドにて行われる
	SampleListener listener;
	Leap::Controller leap;

	// SWIPEとCIRCLEジェスチャーを有効にする
	leap.enableGesture( Leap::Gesture::Type::TYPE_SWIPE );
	leap.enableGesture( Leap::Gesture::Type::TYPE_CIRCLE );

	//リスナー追加
	leap.addListener( listener );

	leap.setPolicyFlags( Leap::Controller::PolicyFlag::POLICY_BACKGROUND_FRAMES ); //バックグラ
ウンドでも起動

	std::cout << "終了するには何かキーを押してください" << std::endl;
	std::cin.get();

	leap.removeListener( listener );
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Leap;

namespace MediaPlayerSampleCS
{
    public partial class Form1 : Form
    {
        SampleListener listener;
        Controller controller;

        public Form1()
        {
            InitializeComponent();

            // Create a sample listener and controller
            listener = new SampleListener(this);
            controller = new Controller();

            // Have the sample listener receive events from the controller
            controller.AddListener(listener);

            controller.SetPolicyFlags(Controller.PolicyFlag.POLICYBACKGROUNDFRAMES); //バックグラウ
ンドで起動
        }

        ~Form1()
        {
            // Remove the sample listener when done
            controller.RemoveListener(listener);
            controller.Dispose();
        }

        private void fileToolStripMenuItem_Click(object sender, EventArgs e)
        {

        }

        private void openToolStripMenuItem_Click(object sender, EventArgs e)
        {
            openFileDialog1.Multiselect = true;

            if (openFileDialog1.ShowDialog() == DialogResult.OK)
            {
                axWindowsMediaPlayer1.URL = openFileDialog1.FileNames[0];
                for (int i = 1; i < openFileDialog1.FileNames.Length; i++)
                {

axWindowsMediaPlayer1.currentPlaylist.appendItem(axWindowsMediaPlayer1.newMedia(openFileDialog1.Fil
eNames[i]));
                }
            }
        }
```

```
        private void axWindowsMediaPlayer1_Enter(object sender, EventArgs e)
        {
        }

        private void openFileDialog1_FileOk(object sender, CancelEventArgs e)
        {
        }

        delegate void PressPlayDelegate();
        delegate void PressStopDelegate();
        delegate void PressBackDelegate();
        delegate void PressNextDelegate();
        delegate void VolumeUpDelegate();
        delegate void VolumeDownDelegate();

        void PressPlay()
        {
            if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPlaying)
            {
                axWindowsMediaPlayer1.Ctlcontrols.pause();
            }
            else if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsStopped
                || axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPaused)
            {
                axWindowsMediaPlayer1.Ctlcontrols.play();
            }
        }
        void PressStop()
        {
            axWindowsMediaPlayer1.Ctlcontrols.stop();
        }
        void PressBack()
        {
            axWindowsMediaPlayer1.Ctlcontrols.previous();
        }
        void PressNext()
        {
            axWindowsMediaPlayer1.Ctlcontrols.next();
        }
        void VolumeUp()

        {
            int volume = axWindowsMediaPlayer1.settings.volume;
            volume = volume + 5;
            if (volume > 100)
            {
                volume = 100;
            }
            axWindowsMediaPlayer1.settings.volume = volume;
        }
        void VolumeDown()
        {
            int volume = axWindowsMediaPlayer1.settings.volume;
            volume = volume - 5;
            if (volume < 0)
            {
                volume = 0;
            }
            axWindowsMediaPlayer1.settings.volume = volume;
        }

        class SampleListener : Listener
        {
```

```
private Form1 form;

private Object thisLock = new Object();

bool mFirst;
Frame mCurrentFrame;
Frame mLastFrame;

List<SwipeGesture> mSwipeGestureList;
List<CircleGesture> mCircleGestureList;

long mPrevLeftSwipeTime; // 以前に左スワイプを検出した時刻
long mPrevRightSwipeTime; // 以前に右スワイプを検出した時刻
long mPrevUpSwipeTime; // 以前に左スワイプを検出した時刻
long mPrevDownSwipeTime; // 以前に右スワイプを検出した時刻

int Progress0 = 0;  //以前のプログレス

public SampleListener(Form1 f)
{
    form = f;

    mFirst = true;
    mSwipeGestureList = new List<SwipeGesture>();
    mCircleGestureList = new List<CircleGesture>();

    mPrevLeftSwipeTime = 0;
    mPrevRightSwipeTime = 0;
    mPrevUpSwipeTime = 0;
    mPrevDownSwipeTime = 0;
}

private void SafeWriteLine(String line)
{
    lock (thisLock)
    {
        Console.WriteLine(line);
    }
}

public override void OnInit(Controller controller)
{
    SafeWriteLine("Initialized");
}

//ジェスチャーの有効化
public override void OnConnect(Controller controller)
{
    SafeWriteLine("Connected");
    controller.EnableGesture(Gesture.GestureType.TYPE_CIRCLE);
    controller.EnableGesture(Gesture.GestureType.TYPE_KEY_TAP);
    controller.EnableGesture(Gesture.GestureType.TYPE_SCREEN_TAP);
    controller.EnableGesture(Gesture.GestureType.TYPE_SWIPE);
}

public override void OnDisconnect(Controller controller)
{
    //Note: not dispatched when running in a debugger.
    SafeWriteLine("Disconnected");
}

public override void OnExit(Controller controller)
{
```

```
                SafeWriteLine("Exited");
            }

            public override void OnFrame(Controller controller)
            {
                if (mFirst)
                {
                    mCurrentFrame = controller.Frame();
                    mFirst = false;
                }

                // フレームの更新
                mLastFrame = mCurrentFrame;
                mCurrentFrame = controller.Frame();

                // 以前のフレームが有効であれば、今回のフレームまでの間に検出したジェスチャーの一覧
を取得する
                // 以前のフレームが有効でなければ、今回のフレームで検出し他ジェスチャーの一覧を取得
する
                GestureList gestures = mLastFrame.IsValid ? mCurrentFrame.Gestures(mLastFrame) :
mCurrentFrame.Gestures();

                // 検出したジェスチャーの履歴を保存する
                foreach (Gesture gesture in gestures)
                {
                    switch (gesture.Type)
                    {
                        // SWIPEジェスチャーをmSwipeGestureListに登録
                        case Gesture.GestureType.TYPE_SWIPE:
                            {
                                SwipeGesture s = new SwipeGesture(gesture);
                                // IDによってジェスチャーを登録する
                                int index = mSwipeGestureList.FindIndex(g => g.Id == gesture.Id);
                                if (index == -1)
                                {
                                    mSwipeGestureList.Add(s);
                                }
                                else
                                {
                                    mSwipeGestureList[index] = s;
                                }
                            }
                            break;
                        // CIRCLEジェスチャーをmCircleGestureListに登録
                        case Gesture.GestureType.TYPE_CIRCLE:
                            {
                                CircleGesture c = new CircleGesture(gesture);
                                // IDによってジェスチャーを登録する
                                int index = mCircleGestureList.FindIndex(g => g.Id == gesture.Id);
                                if (index == -1)
                                {
                                    mCircleGestureList.Add(c);
                                }
                                else
                                {
                                    mCircleGestureList[index] = c;

                                }
                            }
                            break;

                        default:
                            break;
```

```
            }
        }

        bool leftSwipeStopped = false;
        bool rightSwipeStopped = false;
        bool upSwipeStopped = false;
        bool downSwipeStopped = false;
        long leftSwipeTime = 0; // 左スワイプの検出時刻
        long rightSwipeTime = 0; // 右スワイプの検出時刻
        long upSwipeTime = 0; // 上スワイプの検出時刻
        long downSwipeTime = 0; // 下スワイプの検出時刻

        bool clockwiseUpdating = false;
        bool counterClockwiseUpdating = false;

        int Progress; //現在のプログレス

        //これ以内の短い間隔のジェスチャーは検出しない。そうしないとウインドウが大量に起動
する
        long detectionWaitTime = 1000000; // 1000*1000 (μ秒) = 1秒。


        // 検出したSWIPEジェスチャーの履歴を表示する
        foreach (SwipeGesture gesture in mSwipeGestureList)
        {
            if (gesture.State == Leap.Gesture.GestureState.STATE_STOP)
            { // スワイプ停止の検出

                if (gesture.Direction.x <= -0.5)
                {  // 左スワイプの検出
                    leftSwipeStopped = true;
                    leftSwipeTime = gesture.Frame.Timestamp;
                }
                else if (gesture.Direction.x >= 0.5)
                {  //右スワイプの検出
                    rightSwipeStopped = true;
                    rightSwipeTime = gesture.Frame.Timestamp;
                }

                if (gesture.Direction.y >= 0.5)
                {  // 上スワイプの検出
                    upSwipeStopped = true;
                    upSwipeTime = gesture.Frame.Timestamp;
                }
                else if (gesture.Direction.y <= -0.5)
                {  //下スワイプの検出
                    downSwipeStopped = true;
                    downSwipeTime = gesture.Frame.Timestamp;
                }
            }
        }

        // 検出したCIRCLEジェスチャーの履歴を表示する
        foreach (CircleGesture gesture in mCircleGestureList)
        {
            if (gesture.State == Leap.Gesture.GestureState.STATE_UPDATE)
            {//サークル継続検出
                Progress = (int)(gesture.Progress / 0.5);

                if (gesture.Pointable.Direction.AngleTo(gesture.Normal) <= Math.PI / 2)
                {
                    if (Progress != Progress0)
                    {
```

```
                        clockwiseUpdating = true;
                    }
                }
                else
                {
                    if (Progress != Progress0)
                    {
                        counterClockwiseUpdating = true;
                    }
                }
                Progress0 = Progress;
            }
        }

        if (leftSwipeStopped)
        { // 左スワイプが検出されたら (IE)
            if (leftSwipeTime - mPrevLeftSwipeTime > detectionWaitTime)
            {  // 以前の左スワイプ検出より1秒以上たっていたら
                SafeWriteLine("左スワイプ検出");

                form.Invoke(new PressBackDelegate(form.PressBack));

                mPrevLeftSwipeTime = leftSwipeTime;
            }
        }

        if (rightSwipeStopped)
        { // 右スワイプが検出されたら
            if (rightSwipeTime - mPrevRightSwipeTime > detectionWaitTime)
            {  // 以前の右スワイプ検出より1秒以上たっていたら
                SafeWriteLine("右スワイプ検出");

                form.Invoke(new PressNextDelegate(form.PressNext));

                mPrevRightSwipeTime = rightSwipeTime;
            }
        }

        if (upSwipeStopped)
        { // 上スワイプが検出されたら
            if (upSwipeTime - mPrevUpSwipeTime > detectionWaitTime)
            {  // 以前の上スワイプ検出より1秒以上たっていたら
                SafeWriteLine("上スワイプ検出");

                form.Invoke(new PressPlayDelegate(form.PressPlay));

                mPrevUpSwipeTime = upSwipeTime;
            }
        }

        if (downSwipeStopped)
        { // 下スワイプが検出されたら
            if (downSwipeTime - mPrevDownSwipeTime > detectionWaitTime)
            {  // 以前の下スワイプ検出より1秒以上たっていたら
                SafeWriteLine("下スワイプ検出");

                form.Invoke(new PressStopDelegate(form.PressStop));

                mPrevDownSwipeTime = downSwipeTime;
            }
        }

        if (clockwiseUpdating)
```

```
                    { // 時計回りがが検出されたら
                        SafeWriteLine("時計回り検出");

                        form.Invoke(new VolumeUpDelegate(form.VolumeUp));
                    }

                    if (counterClockwiseUpdating)
                    { // 時計回りがが検出されたら
                        SafeWriteLine("反時計回り検出");

                        form.Invoke(new VolumeDownDelegate(form.VolumeDown));
                    }

                    mSwipeGestureList.RemoveAll(g => g.Frame.Timestamp <= mCurrentFrame.Timestamp - (1
* 1000 * 1000));
                    mCircleGestureList.RemoveAll(g => g.Frame.Timestamp <= mCurrentFrame.Timestamp - (1
* 1000 * 1000));

                }
            }// class SampleListener
        }
    }
```