卒　業　論　文

Microsoft Kinect for windows を使用した
モーションキャプチャーによる PC 操作

学　　科	情報システム科
コ ー ス	アプリケーションエンジニア　コース
学籍番号	121Y058
氏　　名	西川　駿

目　　次

1. 要約

昨今、モーションキャプチャーができる機器が一般の PC 向けに続々開発、発売されている。しかし、機器はあってもソフトウェアの開発が追い付いておらず開発者用キットの状態になってしまっており、一般には依然として普及及び認知はされていない。そこで今回モーションキャプチャーができる機器の中でも有名である Microsoft の Kinect を用い、誰でも使うことができるソフトウェアの開発をしてみたいと思った。

今回の研究で使用した言語は C#、理由は Kinect が対応している主な言語が C#と C++であり、C#の方が対応している幅が大きく融通が利くからである。

使用したハード、ソフトウェア、及びプラグインは Microsoft Kinect for Windows、Visual Studio、Kinect for Windows SDK1.8、KinectToolbox(C#のライブラリ)を使用した。Kinect は Xbox 版、Windows 版とあり Windows 版は近くの物を認識できる設定や、座っている時の状態を認識できる設定が有る為、今回 Windows 版の Kinect を使用した。また、ソフトウェアの形式はフォームアプリケーションで開発を行なった。

今回の研究では当初の目的は全て達成する事ができた。要因としては要件定義でプログラムを書いていく順番をしっかり決めた事。

元々知識が無く、大変だと思っていた Kinect for SDK の扱いだが、関数が多く用意してあり、案外簡単に終わった事。一番苦労したモーションを認識する部分の処理がライブラリを解読し、使用する事により解決できた事が大きな要因となっている。

2. 序論

2-1 研究の背景

昨今音声認識、タッチパネルといった従来からある入力機器に加え、加速度センサーやモーションキャプチャ等様々な新しい入力機器が登場している。しかし、後者はまだ機器自体が普及しておらずに一般的にはゲーム機等で使用されているのを見かけるくらいだ。特にモーションキャプチャーについては殆ど一般的な場所では見る事が出来ず、映像を開発する際や医療関係等で主に使われている。こうした中、最近ではモーションキャプチャーの機器が一般の PC 向けに続々開発、発売されている。しかし機器はあってもまだアプリケーションの開発が追い付いておらず開発者用キットの状態になってしまっており、一般には中々普及していない。そこで今回モーションキャプチャーができる機器の中でも有名である Microsoft の Kinect を用い、誰でも使うことができるアプリケーションの開発をしてみたいと思った。

また、プレゼンテーションの際に従来は画面切り替えの為にマウス、キーボード、リモコン等でページ送りをしなければならなかったが、モーションキャプチャを用いれば離れていても操作ができ、よりプレゼンテーションがしやすく、また映えるのではないかと思った。

2-2 研究の目的

Microsoft Kinect for windows のモーションキャプチャーを用いてジェスチャーを認識し、その動きによって PC を操作するアプリケーションを制作するのが主な目的。

機能としては一般的なマウスジェスチャー等と同じブラウザの戻る、進む、タブを開く等の操作をできるようにする。また、ブラウザによりショートカットキーが違う場合もあるので任意のキーを最大三つまで押せるようにする。これによりブラウザだけでなく、PowerPoint 等の他のアプリケーションでも使用できるようになる。

その他にもプロセス毎に設定を変更できるようにし､汎用性を高め使いやすくする。モーションの読み取りの為の数値を変更できるようにし、人に合わせて使えるようにする。

3. 研究の方法

今回の研究で使用した言語は C#。ハード、ソフトウェア及びライブラリは Microsoft Kinect for Windows、Visual Studio、Kinect for WindowsSDK1.8、KinectToolbox(C# のライブラリ)を使用した。

行程はスケジュールを立てる、要件定義、コーディング、テスト、評価の順番で行った。

要件定義では使用する言語を C#に決定し、どの様な構造のプログラムにするか、プログラムをどの順番で書き上げていくかという事を決定した。

コーディング、テストでは要件定義に従い実際に Kinect を使いながら制作、デバッグをした。その後評価をし、使いづらい所はないか、もっと便利にできる所はないか等検討をした。

4. 本論

4-1 本作品の仕様

言語は C#、フォームアプリケーションで作成をした。C#を選んだ理由だが、Kinect で使える言語は C#と C++が主流であり、C#の方が幅広く関数を扱うことができ、また自分も今まで学校で学んできた言語である為今回 C#に決定をした。

また､C#で制作する場合 Kinect のアプリは一般的に WPF アプリケーションを使用して作る場合が多いが、これは画像、動画処理をする場合 WPF アプリケーションの方が作りやすいからであり、今回は画像、動画処理を行わず数値の処理、保存等を主にする為扱いやすいフォームアプリケーショ

ンで制作をした。

本作品はモーションを読み取り、認識後そのモーションに割り当てられているショートカットキーを押す事により PC の操作を可能にしている。モーションは 5 種類認識可能で、下記の通りである。

・SwipeToLeft

左手を左に動かす。

・SwipeToRight

左手を右に動かす・

・R.HandOverHead

右手を頭の上に乗せる。

・L.HandOverHead

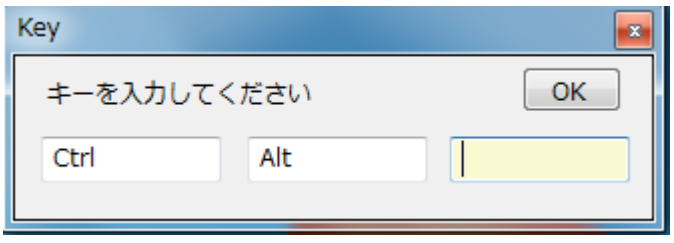
左手を頭の上に乗せる。

・HandsJoint

両手を合わせる。

左手を主に認識するようにした理由だが、右手はマウスは持っている事を前提に製作をしたからであり、あくまでも操作の補助を目的にした。

これらのモーションへの割り当てはデフォルトで"進む・戻る・更新・更新(キャッシュを無視)・ウィンドウを閉じる・タブを閉じる・タブを開く・無効(何もしない)"の 8 種類であり、上記の動作は一般的なブラウザのショートカットキーを割り当てている。しかし、ソフトウェアによってショートカットキーが違っていたり、他の動作もしたい時の場合に自分で任意のキー(最大三つ同時押しまで)を割り当てる事ができる。



ほぼ全てのキーを割り当てる事が出来るが、ピリオド等の極一部の文字を割り当てる事は出来ない。一般的にショートカットキーとして使われるキーは全て割り当てることは出来る。流れとしては入力フォームをクリック(右から)すると色が変わり、そこに入力されていた文字が消える、そしてキーを押すと押したキーが表示されるその後 OK ボタンを押すと反映される。未入力で OK を押した場合は無効(何もしない)と同様の動作になる。

これらの割り当てをプロセス(exe)別に設定する事も可能となっている。これにより同じモーションでもアクティブのウィンドウによって違うショートカットキ

ーを割り当てる事が出来る。これにより様々なアプリケーションで使用する事が可能となる。なお、アクティブウィンドウのプロセス名が登録されていない場合はデフォルトの割り当てを適用する。

プロセスの設定は編集ボタンを押すと起動するエディターにて編集をする。プロセスの追加をする場合はプロセスの追加ボタンを押す、押すとポップアップウィンドウが表示されOKを押すと5秒後にアクティブになっているプロセスの名前を取得、追加する流れになっている。

プロセス1つに付き、

プロセス名

toRight の割り当て

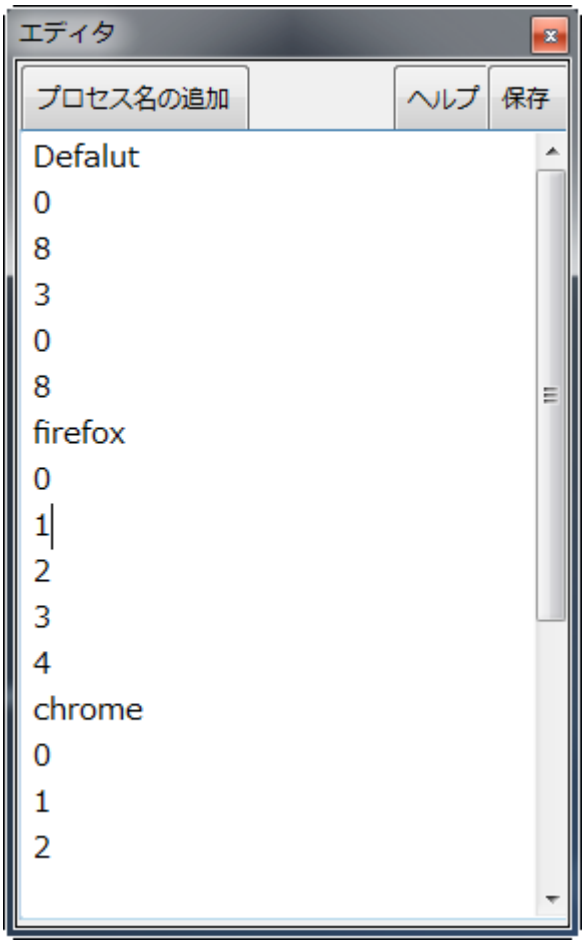
toLeft の割り当て

R.HandOverHead の割り当て

L.HandOverHead の割り当て

HandsJoint の割り当て

の6行で一つの設定になっている。

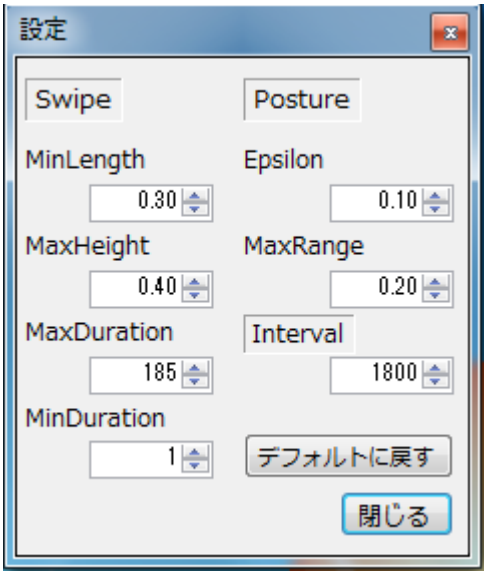


これらのファイルは二次元配列によって起動時に読み込まれており、動作を重くする原因となってしまってもいる。

また、図の用に記述していかずに無駄に改行等を作ってしまうと正しく読み込まれずに動作がおかしくなってしまう。ただし、基本的にこのエディターでの編集はプロセス名の追加のみを行うことを推奨しているため、極端に扱いづらかったりなどはしないと考えられる。(プロセス名の追加と同時に残りの5行も追加される)

エディターを編集後自動で保存はされないので保存ボタンを押してからエディターを閉じる必要がある、エディターを閉じた後に保存したIniファイルを再読み込みして設定を即時反映するようにしている。

モーションの読み取りは環境や扱う人によってどうしても多少の誤作動や上手く認識しなかったりという問題がある。それを少しでも軽減する為にモーション読みとり用の数値を設定、保存できるフォームを実装した。設定項目の詳細は項目名をマウスオーバーする事によって説明が出るようにしている。しかし、数値を変更してテストをしてみても違いがわからずどのような影響を与えるか不明な項目もあり、その説明は"不明"と表示されるようにしか設定していない。



設定項目は主に toRight,toLeft の二つに適応される Swipe タブと R.HandOverHead, L.HandOverHead, HandsJoint の三つに適応される Posture タブ、両方に適応される Interval タブの全部で 3 つのタブで構成されている。

Swipe タブ

・MinLength

スワイプした際の最小移動距離を設定する。数値を大きくすればする程長距離腕を移動させないと認識しなくなるが、数値が小さすぎるとちょっとした動作でも認識してしまい、誤作動してしまう可能性が非常に高くなる

・MaxLength

詳細不明、ライブラリの中身と項目名から察するに移動する際腕が上下にぶれた場合の認識範囲だと思われるがきっちりと検証をすることはできなかった。

・MaxDuration

MinLength で指定した距離を移動させる際の認識する最長時間の項目となっている。短すぎると腕を素早く動かさないと認識しないが、長すぎるとこれも誤動作の原因になってしまうため注意が必要である。

・MinDuration

MaxDuration の反対で MinLength で指定した距離を移動させる際の認識する最短時間となっている。これを"１"以外の数値にすると移動距離が早すぎても認識しないようになる。個人的には必

要性を感じることができなかったが、ライブラリにある項目なので設定をできるようにはした。

Posture タブ

・Epsiilon

詳細不明、位置を認識する際の Z 軸関係かとも予想したが検証をしてみても成果があがらず、どのような項目なのか理解することはできなかった。

・MaxRange

認識する位置の最長判定を設定する。(例：右手の座標と左手の座標の距離)これを大きくする事により、二つの座標が離れていても認識するようになるが誤作動の確立が高くなってしまう。反面あまりにも小さくしすぎてしまうと認識しなくなってしまうので注意が必要である。

Interval タブ

・Interval

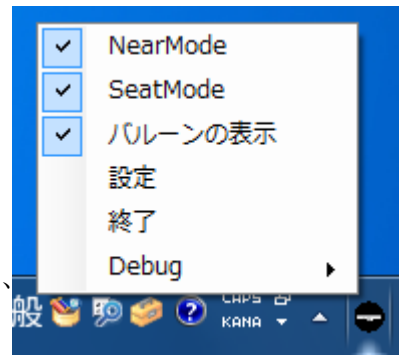
ライブラリの仕様ではそのモーションをしている限りずっと認識しているという信号が送られてくるため連続で動作をしてしまう。それを防ぐ為に今回新たに追加した項目である。一回モーションをすると Interval で設定した秒数(ms)経過するまで新たにモーションを認識しないという仕様になっている。

これらの数値は終了時に自動で保存されるようになっており、特に保存ボタンを押す必要はない。また、数値の設定を間違えてしまい直すのが困難になった場合はリセットボタンを押すと数値を全てリセットし、初期状態に戻す事ができる。

ソフトウェアは起動時に設定フォームが起動するようになっており、その他のエディター、キー設定、モーション数値設定のフォーム等は設定フォームから立ち上げる流れとなっている。

設定フォームはタスクトレイのアイコンから呼び出す事が可能で、Kinect の Mode 設定もタスクトレイのアイコンから変更できるようにした。

アイコン画像は図のようになっており、左クリックでメニューを開く事ができる。



・NearMode

Kinect は通常のモードでは 70cm 以上離れて使用するのが適性距離となっており、

それ以上近くで使用すると正しい座標がとれなくなってしまう事がある。NearMode に設定をすると適性距離が 40cm まで縮める事ができる。細かい座標が取れなくなったり一部の関数が使用できなくなるというデメリットもあるが本作品では使用しない部分なので特に問題は出ない。また、NearMode は XBOX 版の Kinect で使用することはできず、Windows 版の Kinect のみ使用可能な機能である。

・SeatMode

上記同様 Kinect は通常立った状態での使用を想定しており、座った(下半身の座標が取れない)状態で使用をすると正しく座標を取得することができない。SeatMode に設定すると上半身のみの座標を取得するため、椅子に座った状態でも正しく座標を取得する事ができる。上記のモード同様本作品では両方オンの状態で使用する事を想定しているため、起動時には両方オンになっているがもちろん両方切った状態でも正しい立ち位置に居れば正しく座標を取得、動作させる事が可能。

・バルーンの表示

モーションを認識、ショートカットキーを動作させた際にバルーンを表示させるかの有無を設定する、起動時にはオンになっている。これによりどの動作をしたのか明確に判断する事ができる。また、図ではアイコンの変化が無いが実際の動作の際には Interval の間(モーションを認識しない)はアイコンの色が反転し、Interval 中なのかどうなのかを判断しやすいようにしている。

・設定

設定用フォームを開く。

・終了

ソフトウェアを終了させる。ここから終了させずにタスクマネージャー等から強制終了をさせてしまうと Kinect の仕様上 Kinect 本体との接続が正常に終了されず、新たに Kinect を使用するソフトウェアを立ち上げるとエラーが発生してしまう。

・Debug

Debug メニューを開く、基本的に開発用なので説明は省略する。メニューの一つに座標表示をするフォームを立ち上げる事ができ、そこで Kinect が正しく座標を取得できているかを確認することも可能。

起動時に正しくKinectと接続する事ができなかったり、KinectがPCに接続されていない場合はエラーメッセージを出しソフトを終了させるようになっている。

4-2 ソースコードの解説

この章ではソースで主なメソッド、流れを説明する。説明文はコメントアウトでも記載する。

本作品はフォーム6つとライブラリ1つの計7つで構成されている。

・Form1

起動時に読み込まれる最初のフォーム、Kinectのモーションキャプチャー等の処理は全てここで行っている。

・Form2

設定用フォーム、他のフォームのほとんどをここから立ち上げる。

・Form3

開発用に作成したDebug用フォーム、主に座標表示をする。

・Form4

任意のキーを設定する際に開かれるフォーム。

・Form5

プロセス名の追加をする際に開くエディターのフォーム。

・Form6

モーション認識時の数値を設定する際に開かれるフォーム。

・MouseController(ライブラリ)

座標がどのように移動したかを判断し、それに応じてモーションを感知した場合は値を返すライブラリ。

Form1

Form1クラス内のコード

```
private int WM_SYSCOMMAND = 0x112;
private IntPtr SC_MINIMIZE = (IntPtr)0xF020;
//フォームを非表示にさせる為の変数
public bool balloonFlag;
//バルーン表示非表示を変更する為の変数
SwipeGestureDetector swipe;
AlgorithmicPostureDetector posture;
//モーション認識用の変数
KinectSensor kinect;
//Kinectを接続する為の変数
```

```
bool flag;
bool exitFlag;
public bool fm3Flag;
//フラグ処理
Form2 fm2;
//設定用フォームの変数
Form3 fm3;
//座標表示(Debug 用)フォームの変数
private float oldVal;
//座標のブレ確認用の変数
```

Form1

起動時に最初に読み込まれるメソッド

```
private void Form1_Load(object sender, EventArgs e)
{
    oldVal = 0;
    flag = true;
    exitFlag = true;
    fm3Flag = false;
    //フラグ処理
    fm2 = new Form2(this);
    fm2.StartPosition = FormStartPosition.CenterScreen;
    fm2.Show();
    /*
     *設定フォームの宣言をし、最初に表示する際の位置を
     *スクリーンの中心に設定し、設定フォームを表示させる。
     */
    Hide();
    /*
     *このフォームはあくまでもダミーなのでフォームが
     *表示されないように非表示に設定する。
     */
    nMenu.BalloonTipIcon = ToolTipIcon.Info;
    //バルーンを設定する。
```

```
swipe = new SwipeGestureDetector();
posture = new AlgorithmicPostureDetector();
/*
 *モーション認識用の関数を設定、Swipe は toRight､toLeft の
 *二つ用で Posture は残りの三つのモーション用になっている。
 */
nMenu.Visible = true;
/*
 *タスクトレイのアイコンを表示させる。
 */
dummyIcon1.Visible = false;
dummyIcon2.Visible = false;
/*
 *アイコンの色を変化させる為のダミーアイコンを設定、
 *非表示にさせる。
 */
checkKinect();
/*
 *Kinect の接続確認や初期設定をするメソッドを実行。
 */
sheetModeToolStripMenuItem.Checked = true;
PopupMenuToolStripMenuItem.Checked = true;
nearModeToolStripMenuItem.Checked = true;
/*
 *タスクトレイメニューの初期設定をする。
 */
balloonFlag = true;
/*
 *バルーンの表示非表示の初期設定(初期はオン)
 */
Form6 fm6 = new Form6(this);
fm6.Show();
fm6.Close();
/*
 *モーション認識用の数値を読み込むために
 *数値設定用フォームを宣言し、その後数値設定用フォームの
```

```
        *方で数値を設定するため、一度 Show を使い表示をさせて
        *設定をした後にすぐ閉じる。
        */
    }
```

Form1

Kinect の接続確認、及び初期設定用メソッド

```
    private void checkKinect()
    {
        try
        {
            //Kinect が接続されているか確認
            if (KinectSensor.KinectSensors.Count != 0)
            {
                /*
                 *キネクトの座標取得時の設定、ここでは
                 *なるべく数値のブレが少なくなるような
                 *数値に設定している。
                 */
                var parameters = new TransformSmoothParameters
                {
                    Smoothing = 0.0f,
                    Correction = 0.5f,
                    Prediction = 0.5f,
                    JitterRadius = 0.01f,
                    MaxDeviationRadius = 0.01f
                };

                posture.MaxRange = 0.2f;
                posture.Epsilon = 0.1f;
                swipe.SwipeMinimalLength = 0.3f;
                swipe.SwipeMininalDuration = 1;
                swipe.SwipeMaximalDuration = 185;
                MouseController.Current.GlobalSmooth = 20;
```

```
                    /*
                     *上記の設定は基本的に数値設定用フォームから
                     *取得できるようにしているが、ファイルが
                     *存在しなかったり等なんらかの影響で数値が
                     *正しく設定されなかった場合の為に数値を
                     *設定している。
                     */
                    kinect = KinectSensor.KinectSensors[0];
                    /*
                     *kinect に現在接続されている Kinect を入れる。
                     *引数の 0 は一つ目の Kinect という意味である。
                     */
                    kinect.SkeletonStream.Enable(parameters);
                    /*
                     *Kinect の Skeleton(座標)を取得する機能を
                     *オンに設定する。
                     */
                    kinect.Start();
                    //Kinect を起動させる。
                    kinect.DepthStream.Range = DepthRange.Near;
                    kinect.SkeletonStream.EnableTrackingInNearRange =
true;
                    //Kinect を NearMode に設定する。
                    kinect.SkeletonStream.TrackingMode =
SkeletonTrackingMode.Seated;
                    //Kinect を SeatMode に設定する。

                    kinect.AllFramesReady +=
                        new
EventHandler<AllFramesReadyEventArgs>(kinect_AllFramesReady);
                    /*
                     *Kinect の AllFramesReady という Event に
                     *Kinect_AllFramesReady という物を入れる。
                     *このイベントは Kinect がスキャン(毎秒 30FPS)
                     *する度に実行されるイベントである。
                     */
```

```
                }
                else
                {
                    System.Windows.Forms.MessageBox.Show("キネクトが見
つかりませんでした。¥n ソフトを終了します。");
                    exitFlag = false;
                    exit();
                    /*
                     *Kinect が見つからなかった場合は
                     *エラーメッセージを表示して終了させる。
                     */
                }
            }
            catch(SystemException se)
            {
                System.Windows.Forms.MessageBox.Show("不明なエラーが発
生しました。¥n" + se.Message);
                exitFlag = false;
                exit();
                /*
                 *上記の動作中エラーが出て正常に完了しなかった
                 *場合はエラーメッセージを表示して終了させる。
                 *主に Kinect が接続されているが別のソフトウェアで
                 *使用されており使用できなかった際に表示される。
                 */
            }
        }
```

Form1

Kinect の動作 FPS 毎(毎秒 30FPS)に実行されるメソッド

ここの処理が非常に長くなってしまっており、動作が重くなる原因となってしまってもいる。

```
        private void kinect_AllFramesReady(object sender,
AllFramesReadyEventArgs e)
        {
```

```
            try
            {
                //skeletonFrame に取得した Skeleton を入れる。
                using (SkeletonFrame skeletonFrame =
e.OpenSkeletonFrame())
                {
                    int i = 0;
                    /*
                     *skeletonFrame にきちんと座標が入っているかを
                     *確認する、なかった場合は実行しない。
                     */
                    if (skeletonFrame != null)
                    {
                        Skeleton[] skeletons = new
Skeleton[skeletonFrame.SkeletonArrayLength];
                        skeletonFrame.CopySkeletonDataTo(skeletons);
                        //skeletons に取得した座標を入れる。
                        /*
                         *Kinect は複数人の Skeleton を取得する事ができ、
                         *本作品は一人での使用を前提としているが、
                         *途中で座標が取れなくなった場合に skeletons に
                         *入る場所が変わり引数が変わる事があるので、
                         *どこに入っているかを次の For 文で確認する。
                         */
                        for (i = 0; i < skeletons.Length - 1; i++)
                        {
                            Joint check =
skeletons[i].Joints[JointType.Head];
                            //チェック用に頭の座標を取得する。
                            if (check.Position.X != 0)
                            {
                                /*
                                 *0 じゃない(取得できている)場合は
                                 *For 文から脱出し、以後 i を引数と
                                 *して扱う。
                                 */
```

```
                    break;
                }
            }
            Joint wrist =
skeletons[i].Joints[JointType.HandLeft];
            Joint rWrist =
skeletons[i].Joints[JointType.HandRight];
            /*
             *wrist に左手の座標を、rWrist に右手の座標を
             *入れる。実際の動作では現在 rWrist は
             *使用していない。
             */
            posture.TrackPostures(skeletons[i]);
            swipe.Add(wrist.Position, kinect);
            //座標をモーション認識用のライブラリに送る。
            /*
             *Debug メニューの座標表示用のメソッドを
             *実行するかどうかの If 文
             */
            if (fm3Flag)
            {
                /*
                 *Skeleton を座標表示用のフォームに送信し、
                 *フォームで座標を表示させる。
                 */
                fm3.readLabel(skeletons[i]);
            }
            /*
             *Kinect は座標を上手く取れなかったりすると
             *極端に数値がブレてしまいその結果誤作動を
             *起こす事があるのでそれを防ぐために
             *極端な数値変化があった場合はモーションを
             *認識したかどうかのチェックをしないように
             *している。
             */
```

```
                        if (oldVal + 0.3 >
skeletons[i].Joints[JointType.HandLeft].Position.X &&
                            oldVal - 0.3 <
skeletons[i].Joints[JointType.HandLeft].Position.X)
                        {
                            /*
                             *極端な座標のブレが無かった場合は
                             *モーションを認識したかどうかの
                             *チェックをする。
                             */
                            checkGesture(swipe.handGesture);
                            checkGesture(posture.str);
                        }
                        oldVal =
skeletons[i].Joints[JointType.HandLeft].Position.X;
                        /*座標のブレ確認用に今回の座標を入れ、
                         *次回の確認時に使用する。
                         */
                    }

                }

            }

            catch (Exception ex)
            {
                System.Windows.Forms.MessageBox.Show(ex.Message);
                /*
                 *上記の動作で何かエラーが発生し正しく動作を
                 *しなかった場合はエラーメッセージを表示する。
                 */
            }
        }
```

Form1

モーションを認識したかどうかを確認するメソッド

```
        private void checkGesture(string gesture)
        {
            /*
             *この flag は Interval になっており、これが Flase の
             *間は下記の処理を行わない。
             */
            if (flag)
            {
                /*
                 *引数の中身を確認し、それに対応した動作を
                 *させる為の If 文
                 */
                if (gesture == "toRight")
                {
                    flag = false;
                    //Interval 用の Flag を False にする。
                    fm2.goAct(0);
                    //対応したショートカットキーを押す。
                    setIcon(false);
                    //タスクトレイのアイコンの色を反転させる
                    timer1.Start();
                    /*
                     *設定した Interval の長さの Timer を開始させ、
                     *時間が経過した場合は Timer 内で Flag を
                     *True にし、アイコンの色を元に戻す。
                     */
                }
    <--------------------中略---------------------->
                else if (gesture == "RightHandOverHead")
                {
                    flag = false;
                    fm2.goAct(2);
                    setIcon(false);
                    timer1.Start();
                }
```

```
        }
    }
```

Form1
本作品を終了させる際に実行されるメソッド

```
    private void exit()
    {
        /*
         *Kinect が接続されている場合には exitFlag は True に
         *なっており、その時はこの中の物を実行する。
         */
        if (exitFlag)
        {
            kinect.SkeletonStream.Disable();
            kinect.Stop();
            kinect.Dispose();
           /*
            *Kinect の Skeleton(座標)を取得する機能を
            *オフにし、Kinect との接続を切り終了させる。
            */

        }
        fm2.Close();
        fm2.Dispose();
        /*設定用フォームを終了させる。このフォームは
         *実行中の間は Hide されてはいるが常に実行されている。
         */
        this.Close();
        this.Dispose();
        //このフォームを終了させる。
        cMenu.Visible = false;
        /*
         *タスクトレイのアイコンを非表示にする。
         *この処理をしないと終了させてもアイコンが
         *タスクトレイに残ってしまう事がある。
```

```
        */
    }
```

Form2

Form2 を表示した際に最初に読み込まれるメソッド、即ち起動時に読み込まれるメソッドでもある。

```
    private void Form2_Load(object sender, EventArgs e)
    {
        checkBox = new CheckBox[5];
        cb = new ComboBox[5];
        //設定用のボックスを宣言。
        this.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedToolWindow;
        //フォームのレイアウトを設定。
        newText();
        //Ini ファイルが存在するかを確認。
        boxLoading();
        //先ほど宣言したボックスの中身を設定するメソッド
        loadText();
        //Ini ファイルを読み込み、設定をする。
        checkBoxCreate();
        //ボックスの中身に先程読み込んだ設定を入れる。
        createToolTip();
        //マウスオーバーで表示される説明の設定。
    }
```

4-3 モーションによるマウス操作についての考察

当初の予定であれば一通りの機能を実装後、右手でマウスカーソルを動かせる機能を実装しようと思っていたが、実験後それは実装を見送る事に決定した。理由は、Kinect は座標がどの様に動いたかというのを判定するのは得意だが、ある一点の座標を取得するというのが非常に苦手だからだ。どうしても数値にブレ幅が出てしまいカーソルが上手く定まらなかった。クリックを右手を前に出す事で出来るようにしていたがこれも右手を動かしている内に誤作動してしまう事があり、非常に煩わしくなってしまった。また、右手を下げるとカーソルが下がってしまうので常時あげてなければおらず、非常に疲れてしまう。以上の理由からモ

ーションによるマウスカーソル操作は現段階では便利といえず、デメリットが多いので実装を取りやめた。

5. 結論

当初予定していた機能よりも多くの機能を実装する事ができ、動作も概ね良好で当初の目的以上の事は達成できた。しかし、動作をさせる際に背景の影響を受けたり少し立ち位置がずれてしまっただけで座標をとれなくなってしまう、その結果誤作動を起こしてしまう事がありまだまだ改善できる余地はあると感じた。

反省点としては最初の仕様ですべての設定ファイルを Ini ファイルにて保存するようにしたため、プロセス別に設定をする際二次元配列を使用し無理に実装させた為、操作感の低下と CPU への負荷向上、ソースコードの肥大化に繋がってしまった事が唯一後悔している。プロセス別の設定を Ini ファイルではなくデータベースで管理する事で上記の問題を解決出来ると思ったが時間の都合と既にソースコードが二次元配列の処理により複雑に絡み合ってしまっていたので断念をした。

本研究を行なった感想としてはたしかにモーションキャプチャーの技術は進歩しており、一般向けに発売されているものでもそれなりの動作をさせる事ができるが、まだまだ誤差があり誤作動を起こす事もあるのでそれを抑え込むコードを挟んだりする必要があると感じた。しかし他の機器だと mm 単位で指を検知し動かせる物や、ヘッドマウントディスプレイと連動し顔の向きを検知し方向に応じて映像の視点を変える物等様々な物が現在一般向けに開発、発売されているので今後も注視し続けていきたいところである。

6. 参考文献

Kinection.jp

http://kinection.jp/

Kinect の基本知識を参考にした。

7. 謝辞

C#を教えてくださった林先生、藤森先生。本研究をいつも応援してくださった沖先生、本当に感謝しています。有難う御座いました。