Kinectを利用したタッチスクリーンの制作

河内理紗　旗田安純
藤原朱里

１．研究概要

Kinect for Windowsを用いて壁などの平面をタッチスクリーン化し、手の動きを追跡してディスプレイ上にラインを表示させるプログラムを制作した。

２．研究の具体的内容

（１）Kinectとは

2010年にMicrosoft社が発表した家庭用ゲーム機向けの専用インタフェースである。Project Natal という開発コードで呼ばれていたモーションキャプチャの技術を製品化したもので、各種センサーにより体の動き・顔の表情・音声をとらえ、コントローラーを使わずに直感的にゲームをプレイすることができる。(同社のXbox 360、Xbox Oneに対応)

その後、2012年にWindows向けのセンサー「Kinect for Windows」、開発ツールの「Kinect for Windows SDK」が発表され、身振り手振りによって様々な機器を操作することが可能になり、医療や介護、障がい者の支援などの広い分野で活用された。写真１は Kinect for Windows本体である。

写真１　Kinect for Windows本体

（２）開発環境

ソフトウェア：
Microsoft Visual Studio 2015
ライブラリ：Kinect for Windows SDK
言語：Visual C#

（３）タッチスクリーンの制作過程

（ア）セットアップ

Kinect を PC に接続したのち、タッチスクリーンを開始するにあたって諸設定を行う。

・接続確認

Kinect が PC と適切に接続されているかを確認する。図１は、Kinectの接続を確認できなかった場合にはエラーメッセージを表示させ、データが見つからなかった場合には動作を停止させるプログラムである。

```
/// <summary>
/// Kinectの動作を開始する
/// </summary>
/// <param name="kin"></param>
/// <param name="e"></param>
private void StartKinect(KinectSensor kin)
{
    try
    {
        if (KinectSensor.KinectSensors.Count == 0)
        {
            throw new Exception("Kinectが接続されていません");
        }

        kinect = kin;
        kinect.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps30);
        kinect.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
        kinect.AllFramesReady +=
            new EventHandler<AllFramesReadyEventArgs>(kinect_AllFramesReady);

        kinect.Start();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        Close();
    }
}

/// <summary>
/// Kinectの動作を停止する
/// </summary>
/// <param name="kinect"></param>
private void StopKinect(KinectSensor kinect)
{
    if (kinect != null)
    {
        if (kinect.IsRunning)
        {
            kinect.AllFramesReady -= kinect_AllFramesReady;
            kinect.Stop();
            kinect.Dispose();

        }
    }
}
```

図１　Kinectの接続確認

・Kinect のカメラ設定

Kinect のカメラ設定を行う。図２は、Kinect の赤外線センサーで感知した物体と手の距離の値を RGB 画像に変換するプログラムである。

```
/// <summary>
/// RGBカメラ、距離カメラのフレーム更新イベント
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void kinect_AllFramesReady(object sender, AllFramesReadyEventArgs e)
{

    try
    {
        using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
        {
            using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
            {
                if (depthFrame != null && colorFrame != null && kinect.IsRunning)
                {
                    if (depthPixel == null)
                    {
                        depthPixel = new short[depthFrame.PixelDataLength];
                        colorImagePixelPoint =
                            new ColorImagePoint[depthFrame.PixelDataLength];
                    }

                    //描画を3フレームに1回にする
                    if (depthFrame.FrameNumber % 3 != 0)
                        return;

                    depthFrame.CopyPixelDataTo(depthPixel);

                    //距離データの座標をRGB画像の座標に変換する
                    kinect.MapDepthFrameToColorFrame(kinect.DepthStream.Format,
                                                     depthPixel,
                                                     kinect.ColorStream.Format,
                                                     colorImagePixelPoint);

                    //カメラ画像の描画
                    byte[] colorPixel = new byte[colorFrame.PixelDataLength];
                    colorFrame.CopyPixelDataTo(colorPixel);

                    //RGB画像の位置を距離画像の位置に補正
                    colorPixel =
                        CoordinateColorImage(colorImagePixelPoint, colorPixel);

                    CameraImage.Source =
                        BitmapSource.Create(colorFrame.Width, colorFrame.Height,
                        96, 96, PixelFormats.Bgr32, null, colorPixel,
                        colorFrame.Width * colorFrame.BytesPerPixel);
                }
            }
        }
    }
```

図２　Kinect のカメラ設定

・ラインの設定と座標変換

出力するラインの色、太さ、始点・終点の形状、座標変換の設定を行う。座標変換は、領域内の座標をディスプレイ座標に変換するものである。

```
///<summary>
///線を引く
///</summary>
///<param name="start">始点</param>
///<param name="end">終点</param>
public void DrawLine(Point start, Point end)
{
    //座標をディスプレイ座標に変換
    start = ConvertCoordinate(start);
    end = ConvertCoordinate(end);


    // 線を追加
    Line line = new Line();
    line.Stroke = new SolidColorBrush(Color.FromRgb(0, 255, 0));
    line.X1 = start.X;
    line.Y1 = start.Y;
    line.X2 = end.X;
    line.Y2 = end.Y;
    line.StrokeThickness = 30; // 太さ30
    line.StrokeStartLineCap = PenLineCap.Round; // 始点を丸める
    line.StrokeEndLineCap = PenLineCap.Round; // 終点を丸める


    PaintCanvas.Children.Add(line);
}
```

図３　ライン設定

```
/// <summary>
/// 指定した領域座標からディスプレイ座標に変換
/// </summary>
/// <param name="point"></param>
/// <returns></returns>
private Point ConvertCoordinate(Point point)
{
    Point p = new Point();

    p.X = PaintCanvas.Width *
        (1 - (point.X - selectRegion.X) / selectRegion.Width);
    p.Y = PaintCanvas.Height *
        ((point.Y - selectRegion.Y) / selectRegion.Height);

    return p;
}
```

図４　座標変換

（イ） タッチスクリーンの範囲設定

範囲を指定するためには始点になる座標（X,Y）から横幅（Width）・縦幅（Height）の設定が必要である。マウスの始点と終点が必ずしも図５のような対角線でないので、図６の４パターンを設定する。

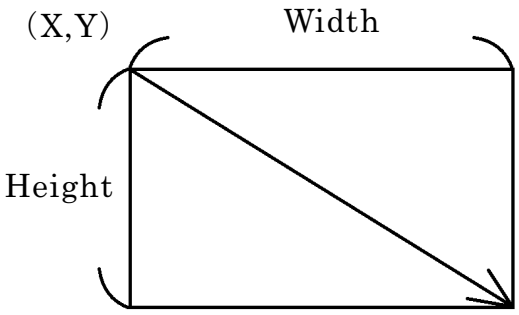

図５　範囲設定の例

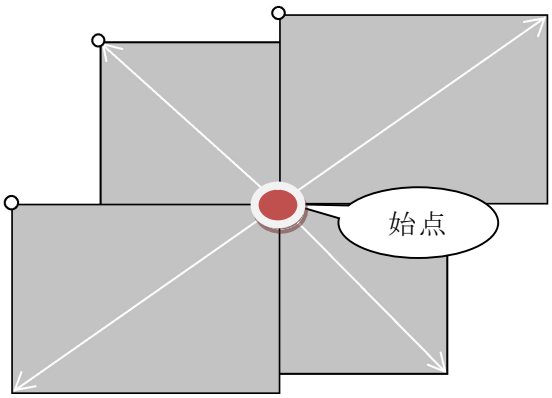

図６　終点検出の４パターン

図７は、マウスの始点の座標と移動した座標の差から縦幅と横幅を計算し、設定するプログラムである。

```
/// <summary>
/// 選択領域を表すRectangleの描画を更新
/// </summary>
private void UpdateRectPosition()
{
    //現在のマウスの位置を取得
    Point currentPoint = Mouse.GetPosition(CameraImage);

    Rect rect;
    //始点と終点の位置によって値を変更
    //
    //終点がスタート位置の左上
    if (currentPoint.X < startPointOfRect.X &&
        currentPoint.Y < startPointOfRect.Y)
    {
        rect = new Rect(currentPoint.X,
                        currentPoint.Y,
                        Math.Abs(startPointOfRect.X - currentPoint.X),
                        Math.Abs(startPointOfRect.Y - currentPoint.Y)
                        );
    }//終点がスタート位置の左下
    else if (currentPoint.X < startPointOfRect.X)
    {
        rect = new Rect(currentPoint.X,
                        startPointOfRect.Y,
                        Math.Abs(startPointOfRect.X - currentPoint.X),
                        Math.Abs(startPointOfRect.Y - currentPoint.Y)
                        );
    }//終点がスタート位置の右上
    else if (currentPoint.Y < startPointOfRect.Y)
    {
        rect = new Rect(startPointOfRect.X,
                        currentPoint.Y,
                        Math.Abs(startPointOfRect.X - currentPoint.X),
                        Math.Abs(startPointOfRect.Y - currentPoint.Y)
                        );
    }//終点がスタート位置の右下
    else
    {
        rect = new Rect(startPointOfRect.X,
                        startPointOfRect.Y,
                        Math.Abs(startPointOfRect.X - currentPoint.X),
                        Math.Abs(startPointOfRect.Y - currentPoint.Y)
                        );
    }

    // Rectangleの配置
    Canvas.SetLeft(SelectRectangle, rect.X);
    Canvas.SetTop(SelectRectangle, rect.X);
    SelectRectangle.Width = rect.Width;
    SelectRectangle.Height = rect.Height;

    //選択領域の保存
    selectRegion = rect;
}
```

図７　範囲設定

写真２のように任意の平面をマウスでドラッグしてタッチスクリーンの範囲を指定する。

写真２　領域の指定

（ウ）タッチスクリーンの実行

指定した範囲内に手などが入ってくると、その部分の距離が変化する。これを利用してタッチ判定や座標の判定を行う。距離カメラから取得する値自体に誤差が含まれるため、その誤差を許容できる程度の余裕を取っておく必要がある。さらに、指定した範囲とKinect との間に人が入った場合に誤認識が起こることを防ぐために、距離変化の変化量が大きすぎる場合も除き、閾値を設ける。

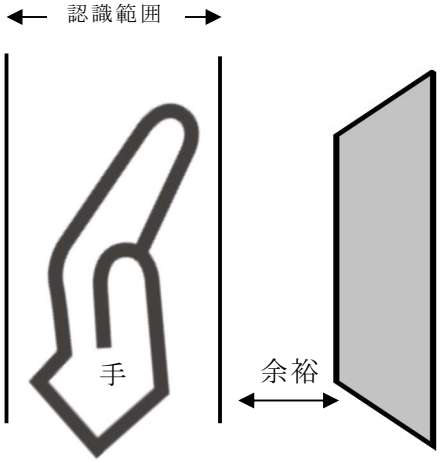

図８　タッチ判定

写真３のように、範囲内でホワイトボードと手が触れているところにカーソル(二重丸)が合い、Paint Window にラインを表示させることができる。

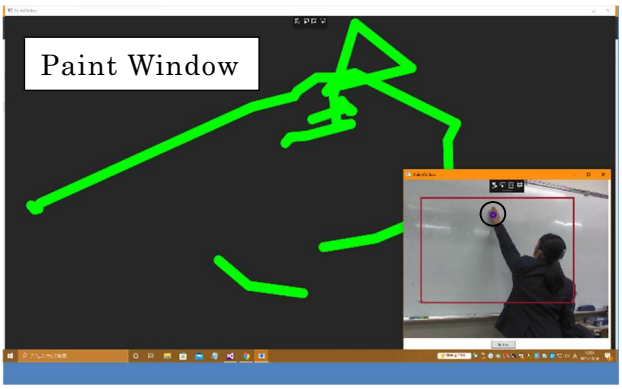

写真３　タッチスクリーン

３．研究のまとめ

課題研究開始当時は Kinect をスポーツの分野において使用する予定であったが、感度が低く認識されづらい、参考資料が古いなどの問題が発生したため断念し、参考文献を頼りにタッチスクリーンの制作へ変更した。最新の Azure Kinect DK ならば感度が上がり、線だけではなく絵や文字も描けるようになると考えられる。会社でのプレゼンや会議だけでなく、学校の授業や講演会などにおいて幅広く活用できるのではないだろうか。

今までに扱ったことがない言語や関数を使用するなかでプログラム内の旧型式に警告が表示され、慣れない操作も多く苦戦したが、及第点に達したと思う。反省点や改善点は多くあるが、課題研究を通して経験した貴重な時間はとても有意義だった。

４．参考文献

KINECT for Windows SDK
プログラミング C#編
著者　中村薫/田中和希/宮城英人