Kinect V2 用いたタッチレスインタフェースの操作に関する研究

A study on touchless interface operation using Kinect V2

余　文爽 [1)]
指導教員　林　誠治 [1)]
1) 拓殖大学大学院　電子システム工学科　林研究室

キーワード：タッチレスインタフェース、スケルトン情報、ジェスチャー操作、プログラミング、C#

1. はじめに

Kinect には RGB カメラ、深度センサー、アレイマイク、および専用ソフトウェアを動作させるプロセッサを内蔵したセンサーがあり、プレイヤーの位置、動き、声、顔を認識することができる。これにより、プレイヤーは自分自身の体を使って、直観的に操作することができる。例えば、リビングルーム、室外等のデバイスを使うことができない場所でパソコンを使いたい場合は、画面操作といったジェスチャー操作での運用が可能である。

2. マイクロソフト Kinect V2

2.1. Kinect V2 について

Kinect V2 とは Kinect V1 を持つ機能の上に、センサーはより高精度となり、カラー画像解像度、Depth 解像度の向上、骨格を追跡できる人数および関節数の増加が強化されている。また、SDK はより簡単に、より高性能になり、顔の表情や状況を取得することができるハードウェアである。

2.2. Kinect V2 の主なスペック

V2 の主なスペックとしては、カラー画像解像度 1920×1080 であり、人の骨格検出 25 点/人で、図 1 に示すように、親指と指先と首ポイントがスケルトンデータに追加されている。

3. 本研究での開発環境

3.1. 準備

Windows SDK 2.0 には Windows8.1 以上の OS が必要ため、Windows 10 をインストールした PC に、Kinect for Windows SDK 2.0 をインストールした。

3.2. サンプルプログラム

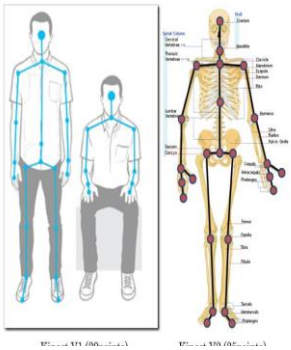

図 1: スケルトンデータ

Kinect for Windows SDK 2.0 のサンプルプログラム“body index”は、手の状態を検出し、3 つのモーションに対して、それぞれの色で判断するプログラムである。

3.3. 本研究での手の状態の追跡システム

3.3.1. 各種座標系と座標マッパクラス

座標マッパ（CoordinateMapper クラス）では各センサーから取得したデータを相互に変換する機能であり、図 2 に示すように、カラー座標系、Depth 座標系、カメラ座標系の三種類に分けられる。

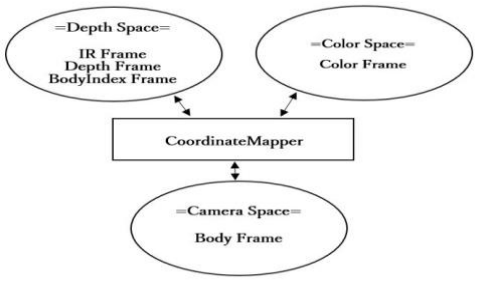

図 2: 各種座標系と座標マッパクラス

カラー座標系は、左上を原点とした 1920×1080 ピクセルの 2 次元座標であり、使うストリームは Color である。Depth(深度) 座標系は、左上を原点とした 512×424 ピクセルの 2 次元座標であり、使うストリームは Depth、Infrared、BodyIndex である。カメラ座標系は、Kinect を原点としたメートル単位での 3 次元座標であり、使うストリームは Body である。ここでのストリームとは、連続したデータの流れとデータの送受信や処理を連続的に行うことを示す。

3.3.2. 手の検出

Kinect V2 では、手の状態は 2 人まで検出できる。また、状態は HandState、手の状態の信頼性 TrackingConfidence で表されている。右手または左手が追跡状態であったときに、それぞれの手の状態を表示している。このとき HandRight (Left) Confidence と HandRight (Left)State を合わせて取得できる。HandState や TrackingConfidence は表 1 のように定義されている。

表 1: 手の状態の定義

値	意味
Open	パーの形
Lasso	チョキの形
Closed	グーの形
High	手の状態の信頼性が高い
Low	手の状態の信頼性が低い

4. プログラム解析

4.1. Main window

本プログラムのメインウィンドウは、図 3 に示すように、初めに Kinect sensor を取得する。そして Multi Source Frame Arrived イベントハンドラの登録を行い、ウィンドウが閉じられるまで非同期にフレーム読み込みが行われる。

4.2. イベントハンドラ

本プログラム処理部では、図 4 の示すように最初にカーソル安定化や初期設定等を行い、カラーフレーム情報取得し画面に映す、そして人を検出するまで待ち、人を認識した場合は、kinect からのカラー画像や depth 画像から取得した情報を利用して、対象の外形（身長）によってカメラ座標系に関節ポイントや骨格モデルを形成し、対象の動きによって追跡することができる。本プログラムでは、対象を認識した時点で、直接カメラ座標系から body データを取得する。その中から椎骨や両手のデータを取り出し、両手や親指の座標をカラー座標に変換して、画面に両手や親指の位置を描画する。続いてスクリーンの高さや幅の長さ、現在カーソル座標を取得し、手の座標と椎骨座標で新たなカーソル座標を計算したのち、実際のマウスカーソルを移動する。

4.3. カーソル座標の計算

新たなカーソル座標は、以下の 2 つの計算式で計算する。まず、(手座標)－(椎骨座標)でフロート型の仮定座標を作り、その後、横軸はプラス 0.05 で校正し、縦軸は椎骨が低いためプラス 0.51 で校正することで頭の高さを適切に調整する。

仮定値 x = handRight.x - spineBase.x + 0.05f
仮定値 y = handRight.y - spineBase.y + 0.51f

仮定座標において、平滑化係数やカーソル感度係数などユーザが調整できる項目を設け、以下の計算式により実際のマウスカーソル座標を導く。

最終座標 SetCursorPosX = curPos.X + (x × mouseSensitivity × screenWidth - curPos.X) × smoothing

右辺は現在カーソル座標+（仮定値×マウス感度×スクリーン幅 - 現在のカーソル座標）×平滑化係数である。

5. 実行結果と考察

5.1. 実行結果

図 5 の示すように、手および親指の位置を認識でき、クリック、ドラッグ・アンド・ドロップなどのカーソル操作を行うことが可能となった。

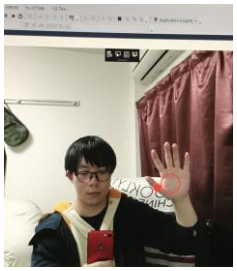

図 5: 実行結果

5.2. 考察

今回プログラム動作の確認を通して、手と親指の認識およびカーソルの動作を確認できたが、手の位置の歪みや操作の精度が低いという2つの問題点が浮上した。解決策としては、手の位置の問題は情報を取得する方法の改善、あるいは表示部 canvas 上でのオフセット値の修正で解決可能であると考えられる。そして操作精度の問題は、計算式の各パラメータの最適化によって改善を試みるなどが考えられる。

6. 今後の予定

今後の予定は、問題点を解決するとともに、マウスの右クリックなど機能の追加、Gesture builder の利用によって、両手を組み合わせたジェスチャー操作機能の開発を目指す予定である。

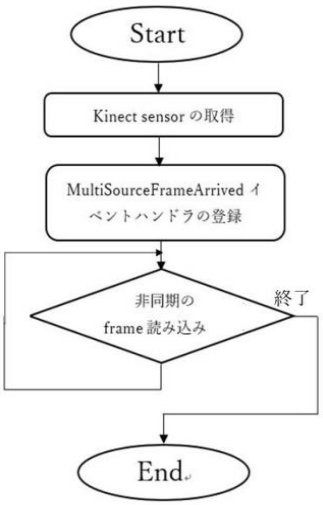

図 3: Main window のフローチャート

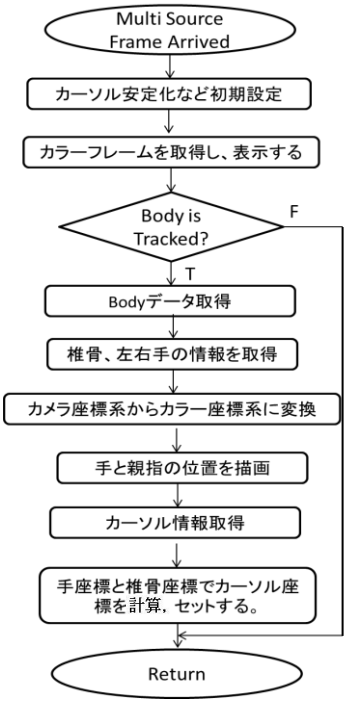

図 4: カーソル座標の計算フローチャート

参考文献

[1] “Kinect Wiki”, http://ja.wikipedia.org/wiki/Kinect
[2] “tlhiv”, http://www.tlhiv.org/rast2vec/
[3] 中村薫ほか, “KINECT forWindows SDK プログラミング”, 秀和システム, 2015 年
[4] “microsoft Developer Network” https://msdn.microsoft.com/ja-jp/