Processing ではじめる Kinect プログラミング

応用

第2回 ジェスチャ操作インタフェースを作ろう

橋本 直（（独）科学技術振興機構）

身体動作の計測に挑戦

前回は Processing で Kinect プログラミングをするための準備といろいろな画像データの取得方法について解説しました．奥行き方向の距離計測や人物領域の抽出など，普通のカメラでは難しかったことが Kinect を使えば驚くほど簡単にできてしまうことがお分かりいただけたと思います．今回は Kinect の醍醐味とも言える，人の身体動作を計測する方法について解説します．まず，人の骨格を認識してトラッキングする方法を説明し，次に手や脚などの関節位置を取得する方法について説明します．最後にその応用例として，3D モデルをジェスチャ入力によって操作するインタフェースの作り方を紹介します．

骨格の認識とトラッキング

Kinect を使って人の骨格を認識し，トラッキングするプログラムをリスト 1 に示します．画面内にユーザが入ってくると，自動的に姿勢を認識して骨格を重畳表示します（図 -1）．

プログラムの中身について解説します．onNewUser() と onEndCalibration() は simple-openni で定義されているコールバック関数です．onNewUser() は新しいユーザが検出されたときに呼び出され，onEndCalibration() はユーザに対する骨格のキャリブレーションが終了したときに呼び出されます．ほかにもこのようなコールバック関数がありますが，ここでは必要最小限のものだけを使用しています．処理の流れはこうです．まず，setup() においてミラー反転，深度画像の取得，カラー画像の取得，ユーザトラッキング，視点の位置合わせなどの機能を有効化します．setup() の処理が終わると draw() を繰り返し実行するループに入りますが，それと並行して Kinect に関するコールバック処理も行われるようになります．画面内に新しいユーザがいることが検出されると，onNewUser() が自動的に呼び出されます．onNewUser() の中では，検出されたユーザに対する骨格のキャリブレーションをリクエストしています．そのキャリブレーションが終了すると，onEndCalibration() が自動的に呼び出されます．

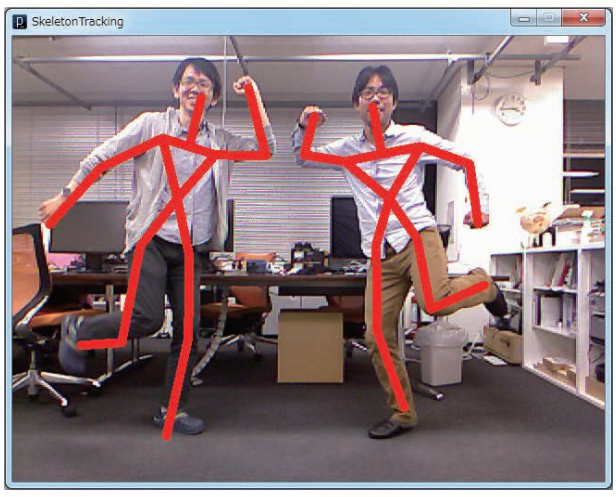

図 -1　骨格の認識とトラッキング

［リスト 1　骨格の認識とトラッキングを行うプログラム］

```
import SimpleOpenNI.*;  // simple-openni

SimpleOpenNI kinect;

void setup() {
  kinect = new SimpleOpenNI(this);
  kinect.setMirror(true);                                    // ミラー反転を有効化
  kinect.enableDepth();                                      // 深度画像を有効化
  kinect.enableRGB();                                        // カラー画像を有効化
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);          // ユーザトラッキングを有効化
  kinect.alternativeViewPointDepthToImage();                 // 視点の位置合わせ
  size(kinect.rgbWidth(), kinect.rgbHeight());               // 画面サイズの設定
}

void draw() {
  kinect.update();                     // Kinect のデータの更新
  image(kinect.rgbImage(), 0, 0);      // カラー画像の描画

  // ユーザごとに骨格のトラッキングができていたら骨格を描画
  for (int userId = 1; userId <= kinect.getNumberOfUsers(); userId++) {
    if( kinect.isTrackingSkeleton(userId) ) {
      strokeWeight(10);        // 線の太さの設定
      stroke(255,0,0);         // 線の色の設定
      drawSkeleton(userId);    // 骨格の描画
      detectGesture(userId);   // ジェスチャ認識
    }
  }
}

// 新しいユーザを見つけた場合の処理
void onNewUser(int userId) {
  kinect.requestCalibrationSkeleton(userId, true);   // 骨格キャリブレーションのリクエスト
}

// 骨格キャリブレーション終了時の処理
void onEndCalibration(int userId, boolean successfull) {
  if (successfull) {
    kinect.startTrackingSkeleton(userId);   // 骨格トラッキングの開始
  }
}

// 骨格の描画
void drawSkeleton(int userId) {
  // 関節間を結ぶ直線を描画
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD,              SimpleOpenNI.SKEL_NECK);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,              SimpleOpenNI.SKEL_LEFT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,     SimpleOpenNI.SKEL_LEFT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,        SimpleOpenNI.SKEL_LEFT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,              SimpleOpenNI.SKEL_RIGHT_SHOULDER);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,    SimpleOpenNI.SKEL_RIGHT_ELBOW);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,       SimpleOpenNI.SKEL_RIGHT_HAND);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,     SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,    SimpleOpenNI.SKEL_TORSO);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,             SimpleOpenNI.SKEL_LEFT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,          SimpleOpenNI.SKEL_LEFT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,         SimpleOpenNI.SKEL_LEFT_FOOT);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,             SimpleOpenNI.SKEL_RIGHT_HIP);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,         SimpleOpenNI.SKEL_RIGHT_KNEE);
  kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,        SimpleOpenNI.SKEL_RIGHT_FOOT);
}

// ジェスチャ認識
void detectGesture(int userId) {
  /* あとでここにジェスチャ認識のコードを書きます */
}
```

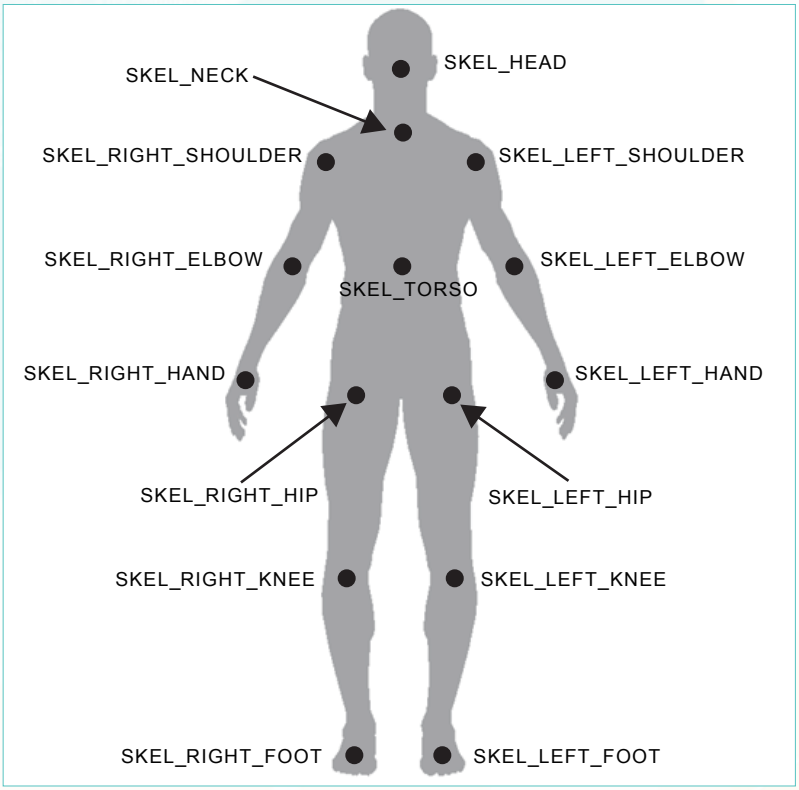

図 -2　関節を表す定数

onEndCalibration() の中では，キャリブレーションの成否を判定後，ユーザに対する骨格のトラッキングを開始しています．draw() の中では，ユーザごとに骨格のトラッキングの可否をチェックし，トラッキングできていれば drawSkelton() という自作の関数によって骨格の描画を行っています．drawSkelton() の中では，drawLimb() というメソッドを使って関節間を結ぶ直線を描画しています．最初の行で使われている SKEL_HEAD と SKEL_NECK は，それぞれ頭と首を意味する定数です．simple-openni では，15 点の関節情報を得ることができます．関節を表す定数を図 -2 に示します．

挙手を認識させてみよう

さきほどのプログラムをベースに簡単なジェスチャ認識を組み込んでみましょう．リスト 1 の detectGesture() の中にジェスチャ認識に関するコードを書き加えます．ここでは「右手を挙げた」という状態を認識させてみることにします．右手が挙がっているかどうかを判定し，挙がっていれば右手に円を表示する処理をリスト 2 に示します．実行結果は図 -3 のようになります．

simple-openni では，関節ごとに空間中の 3 次元位置と画面上の 2 次元位置を取得することができます． getJointPositionSkeleton() が指定された関節の 3 次元位置を取得するメソッドです．さきほど骨格を表示させた時と同様に，図 -2 の定数群を使って関節を指定します．このメソッドは PVector 型のデータを返します．PVector 型の変数からは，hand3d.x，hand3d.y，hand3d.z のようにして xyz 各方向の座標値を得ることができます．また，convertRealWorldToProjective() というメソッドを使えば，空間中の 3 次元の座標値を，それに対応する画面上の 2 次元の座標値（x,y）に変換することができます．リスト 2 では，これら 2 つのメソッドを駆使して右手と右肩の 3 次元位置を取得し，鉛直方向（y 方向）の座標値をチェックして，右手が右肩よりも高い位置にあると判定されたときに，画面上の右手の位置に円（ellipse）を表示するという処理を行っています．

［リスト 2　右手が挙がっていることを認識する処理］

```
void detectGesture(int userId) {
  // 右手の 3 次元位置を取得する
  PVector hand3d = new PVector(); // 3 次元位置
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_HAND, hand3d);

  // 右手の 2 次元位置を取得する
  PVector hand2d = new PVector(); // 2 次元位置
  kinect.convertRealWorldToProjective(hand3d, hand2d);

  // 右肩の 3 次元位置を取得する
  PVector shoulder3d = new PVector();
  kinect.getJointPositionSkeleton(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER, shoulder3d);

  // 右手が右肩よりも高い位置にあるとき，手の 2 次元位置に円を表示する
  if ( hand3d.y - shoulder3d.y > 0 ) {
    ellipse( hand2d.x, hand2d.y, 50, 50 );
  }
}
```

3D モデルをジェスチャ操作するインタフェースを作ろう

Kinect を使って手や脚の関節位置を取得する方法が分かりましたので，その応用としてジェスチャ入力のインタフェースを作ってみましょう．画面に表示された 3D モデルをハンドジェスチャによって回転操作するプログラムをリスト 3 に示します．このプログラムでは，一般的な 3D モデリングツールが対応している OBJ 形式の 3D モデルを描画しています．Processing で OBJ 形式の 3D モデルを扱えるようにするには，OBJLoader というライブラリをインストールする必要があります．

saitoobjloader - Processing OBJLoader library
http://code.google.com/p/saitoobjloader/

上記サイトからライブラリをダウンロードし，simple-openni をインストールしたときと同様に，「libraries」フォルダ内に「OBJLoader」フォルダを入れてください．ここで使用している 3D モデルは OBJLoader のサンプルの中に同梱されているものです．自分が作ったプログラムの pde ファイルが保存されているローカルディレクトリ内に「data」という名前のフォルダを作り，その中に cassini.obj と cassini.mtl をコピーしてください．

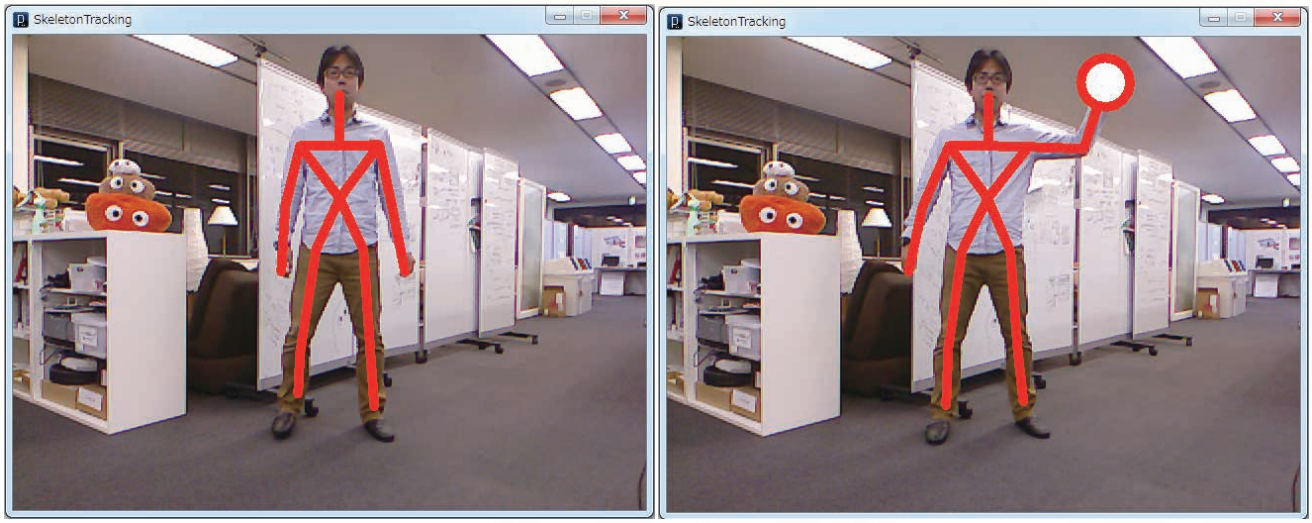

図 -3　右手が挙がっていたら手の上に円を表示（ミラー反転しているので画像上は左右が逆です）

［リスト 3　3D モデルをジェスチャで操作するプログラム］

```
import SimpleOpenNI.*;        // simple-openni
import saito.objloader.*;     // OBJ ローダ

SimpleOpenNI kinect;          // Kinect
OBJModel     model;           // 3D モデル
int          cntlUser = 0;    // 操作を行うユーザの ID
float        rotX;            // 3D モデルの回転角度（X 軸）
float        rotY;            // 3D モデルの回転角度（Y 軸）

void setup() {
  size(640, 480, P3D);  // 画面のサイズ設定（3D 空間を扱うので P3D を付けます）

  // 3D モデルのロード
  model = new OBJModel(this, "cassini.obj", "relative", TRIANGLES);
  model.scale(8);
  noStroke();

  // Kinect に関する設定
  kinect = new SimpleOpenNI(this);
  kinect.setMirror(true);
  kinect.enableDepth();
  kinect.enableRGB();
  kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);
  kinect.alternativeViewPointDepthToImage();
}

void draw() {
  background(32);                              // 背景のリセット
  kinect.update();                             // Kinect のデータ更新
  image(kinect.rgbImage(), 0, 0, 160, 120);    // 画面左上に小さくカラー画像を表示

  // 骨格がトラッキングできていたら 3D モデルの操作を行う
  if ( kinect.isTrackingSkeleton(cntlUser) ) {
    // 右手と右肩の 3 次元位置を取得
    PVector hand = new PVector();
    PVector shoulder = new PVector();
    kinect.getJointPositionSkeleton(cntlUser, SimpleOpenNI.SKEL_RIGHT_HAND, hand);
    kinect.getJointPositionSkeleton(cntlUser, SimpleOpenNI.SKEL_RIGHT_SHOULDER, shoulder);

    // 手と肩の距離から 3D モデルの回転量を決定
    rotY += (hand.x - shoulder.x) * 0.0006;  // 左右方向
    rotX += (hand.y - shoulder.y) * 0.0006;  // 上下方向
  }

  lights();                              // 照明の設定
  translate(width/2, height/2, 0);       // 画面中央に配置
  rotateX(rotX);                         // X 軸まわりの回転
  rotateY(rotY);                         // Y 軸まわりの回転
  model.draw();                          // 3D モデルの描画
}

// 新しいユーザを見つけた場合の処理
void onNewUser(int userId) {
  kinect.requestCalibrationSkeleton(userId, true);  // 骨格キャリブレーションを行う
}

// キャリブレーション終了時の処理
void onEndCalibration(int userId, boolean successfull) {
  if (successfull) {
    kinect.startTrackingSkeleton(userId);   // 骨格トラッキングを開始
    cntlUser = userId;                      // 操作を行うユーザを決定
  }
}
```

このプログラムでは，右手を上下左右に動かすことによって 3D モデルの回転方向を操ることができます（図 -4）．アルゴリズムはとてもシンプルです．肩の位置を基準とし，x 方向・y 方向それぞれで肩から手までの距離に比例した回転量を 3D モデルに与えています．手を動かした方向に 3D モデルが回転し，手をまっすぐ正面に突き出しているときは回転が止まります．仕組みは単純ですが，実際に体験してみるとあたかも魔法でモノを操っているような気分になれます．

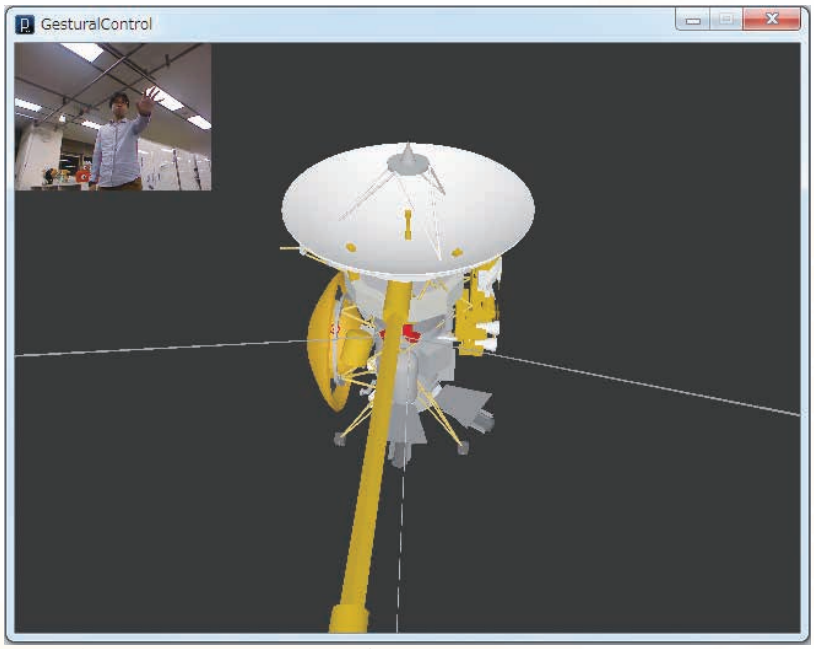

図 -4　ジェスチャ入力によって 3D モデルを操作する

ジェスチャ操作インタフェースの勘所

Kinect を使ったプログラミングはいかがでしたか？　手に何も持たずにジェスチャのみによって操作するアプリケーションがこれほど手軽に作れてしまうのは大きな魅力です．ジェスチャ操作インタフェースを実際に自分で作ってみるといろいろなことに気付かされます．たとえば，完成したプログラムを家族や友人に体験させてみてください．子供やお年寄りでもきちんと動作するでしょうか？　プログラムの中に「手と肩の距離が 100 以上あったら」のように絶対的な数値を含んだ判定処理を書いていたらうまく動作しないかもしれません．また，デバッグ中に何度も腕を上げ下げしていると次第に疲れてきてしまいます．作った人が疲れるのであれば，当然，実際にユーザが使う場合にも同じことが起きるでしょう．疲れないジェスチャとはどのようなものでしょうか？　ぜひ考えてみてください．これ以外にも，意識的な動作とふいにやってしまう無意識の動作の区別や，不正確な入力が与えられた際のフィードバックなど，いろいろな課題が見えてきます．この辺りのことをどうやって解決していくかが，今後ジェスチャ操作インタフェースが世の中に普及していくためのカギであり，研究として面白くなっていく部分だと思います．Kinect を使ったプログラミングを通して，この技術が作る未来の生活について思いを馳せていただければ幸いです．

（2012 年 6 月 11 日受付）

橋本 直（正会員）｜hashimoto@designinterface.jp

2009 年九州工業大学大学院工学研究科博士後期課程修了．博士（工学）．同年より（独）科学技術振興機構 ERATO 五十嵐デザインインタフェースプロジェクト研究員．人とロボットのインタフェースに関する研究に従事．