Kinectを用いた画像処理プログラミング(1)

my recommendations on research & development tools.

松井 勇佑[†]

キーワード: Kinect, OpenCV, 画像処理

1. ま え が き

本稿では2011年度冬学期に東京大学にて行われた「電子情報ゼミ: Kinectを用いた画像処理プログラミング」という少人数講義での取組みについて解説します。講義は受講者5人の少人数ゼミ形式講義として行われ、受講対象者は学部1, 2年生で全員がプログラミング未経験者でした。

講義は前半4回を相澤研究室、後半4回を佐藤研究室にて行い、最後に発表会を行いました。前半ではKinectを使う設定、画像処理プログラミングの基礎、デプスを用いた処理を学び、後半はスケルトンを用いた処理、作品制作を行い、最後に発表会を行いました。本稿では前半について解説します。来月号では後半部分を解説します。

従来、「人間の動作を取得し処理を施す」ことは簡単に行える作業ではありませんでした。しかし近年、安価なデプスカメラであるKinectの登場、およびOpenCVに代表されるコンピュータビジョンライブラリーの発展により、誰でも簡単にNUIプログラミングを行える時代が到来しつつあります。本講義では、プログラミング未経験者だった受講者が3ヵ月で3次元画像処理や音声処理を含むインタラクティブ作品を作り上げることができました。本稿が今後のKinectを用いた画像処理講義の参考となれば幸いです。

2. 環 境

本講義で用いた環境について述べます。

- ・デバイス: Kinect for Xbox 360
- ・SDK: Kinect for Windows SDK Beta
- ・CVライブラリー: OpenCV 2.3.1
- ・ラッパー: 中村薫氏のラッパー¹⁾
- ・OS: Windows7
- ・IDE: Visual Studio 2010
- ・言語: C++

各受講者にノートPCを貸与し、上記の環境を構築しました。2011年9月当時、ドライバとしてMicrosoft公式のものとしてOpenNIを用いるものが、二つの大きな流れとし

てありました。OpenCVは、OpenNIをサポートしていることもあり、環境の選択には頭を悩ませましたが、Microsoft公式のものを選択しました。また言語としては、OpenCVとの連携をスムーズに行うため、C++を選択しました。SDK BetaのC++インタフェースは、ポインタ操作を介すなど初学者には難しい面があったので、中村薫氏により公開されているラッパーを用い、実装の詳細は隠蔽して使用しました。画像の表示や画像処理にはOpenCVを用いました。Kinect関連技術は進展が速く、この原稿の執筆時にはすでに新しいデバイスとしてKinect for Windows SDKとしてKinect for Windows SDK 1.5が入手可能であり、近距離モード、顔のトラッキング、上半身の認識など、新しい機能が次々にサポートされています。常に新しい情報源に当たると良いと思います。

3.1 講義初日: 環境構築, 画像表示

まずは、画像処理の概略、および、Kinectの動作原理について、相澤教授による解説がありました。これにより、なぜKinectを用いると物体までの距離を測定できるのか解説されました。

続いて、ソフトウェア式のインストールを行いました。本講義では、一切の構築を講義中に行おうという方針でしたので、PCは素の状態で配り、Visual Studio, SDK, ラッパー, OpenCVと順に入れていきました。SDKそのものは簡単にインストールでき、サンプルアプリケーションを動かすことで、まずはKinectに触れることができます。OpenCVの導入およびVisual Studioの設定には、予定していたよりずっと時間がかかりましたので、ある程度の環境は構築してからPCを配布したほうが良いかもしれません。

構築が終わると、実際にプログラミングに移ってもらいました。プログラミングは、課題資料を制作・配布し、サンプルプログラムを書いてもらい、途中の課題を解いてもらいました。初日ではまず、画像を単純に表示するサンプルを書いてもらいました。

3.2 講義二日目: KinectからRGB画像とデプス画像を表示

二日目は、画像・デプスの表示を行いました。RGB画像もデプスも同じように配列上の値として表現されることや、カメラとIRカメラの位置の違いから、画像とデプス画

[†] 東京大学 大学院学際情報学府

"Image Processing using Kinect (1)" by Yusuke Matsui (The University of Tokyo, Tokyo)

像がずれることなどを解説しました。

3.3 講義三日目：光学迷彩

講義前半の総集編として、光学迷彩的な表現をプログラミングしてもらいました。そのコードをソースコード1に示します。ここで光学迷彩的な表現とは、図1のようなカメラ入力に対し、図2のデプスキーを用いて、図3の背景を合成し、図4のように透明人間になったかのような映像を得ることを指します。単に背景を重ねるのではなく、デプスの値に応じて上下にずれた位置の背景画素を重畳しています。背景は最初に一度撮影します。処理はリアルタイムで行われるので、カメラの前で人物が動くとまさに透明人間となり動いているように感じます。Kinectに近付き

```
#include <iostream>
using namespace std;

#include "kinect\nui\Kinect.h"
#include "kinect\nui\ImageFrame.h"
using namespace kinect::nui;

#include "kinect_utility.h"

#include <opencv2/opencv.hpp>
using namespace cv;

// Depthの範囲を与える
#define DEPTH_MIN 6408.0
#define DEPTH_MAX 31800.0

int main(int argc, char *argv[])
{
    // kinectクラスの宣言と初期化
    Kinect kinect;
    kinect.Initialize(NUI_INITIALIZE_FLAG_USES_COLOR |
        NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX);

    // ストリーム作る。画像用とデプス用
    ImageStream& video = kinect.VideoStream();
    video.Open( NUI_IMAGE_TYPE_COLOR,
        NUI_IMAGE_RESOLUTION_640x480 );

    ImageStream& depth = kinect.DepthStream();
    depth.Open( NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX,
        NUI_IMAGE_RESOLUTION_320x240 );

    // opencvのmatとwindowの準備
    Mat camera_img = Mat(Size(video.Width(), video.Height()),
        CV_8UC4);
    Mat depth_img = Mat(Size(depth.Width(), depth.Height()),
        CV_16UC1);
    Mat adjust_img = Mat(depth_img.size(), CV_8UC4); //カメラ画像
    // デプス画像に微調整するために使う
    Mat background_img = Mat(camera_img.size(), CV_8UC4); //キャプ
    // チャした背景を保存するために使う
    Mat dst_img = Mat(depth_img.size(), CV_8UC4); //最終結果表
    // 示用に使う
    namedWindow("depth_window");
    namedWindow("adjust_window");
    namedWindow("background_window");
    namedWindow("dst_window");
```

プログラム1 光学迷彩

```
// カメラ画像をデプス画像に微調整するためのエンジン
ColorFromDepthEngine CFDengine;

while ( 1 ) {
    // データの更新を待つ
    kinect.WaitAndUpdateAll();

    // 次のフレームのデータを取得する
    ImageFrame image( video ); // カメラから画像
    DepthFrame depthMD( depth ); // デプスカメラからデプス画像

    // データのコピー
    camera_img = Mat(camera_img.size(), CV_8UC4, (BYTE
        *)image.Bits());
    depth_img = Mat(depth_img.size(), CV_16UC1, (BYTE
        *)depthMD.Bits());

    // depthに合わせて微調整した画像の取得(サイズはデプスに合うよう
    // に小さくなっている)
    adjust_img = CFDengine.getAdjustedImage(camera_img);

    // 以下が光学迷彩のアルゴリズム
    dst_img = adjust_img.clone(); // 微調整済みカメラ画像を結果表示画
    // 像にコピー
    for(int j=0; j<depth_img.rows; j++){
        for(int i=0; i<depth_img.cols; i++){
            short depth_value = depth_img.at<ushort>(Point(i, j)); // 注目点
            // のdepthの値を取り出す
            if(depth_value != 0){
                double rate = (DEPTH_MAX - depth_value) / (DEPTH_MAX -
                    DEPTH_MIN); // 近いほど値大きい(0から1)
                int decreased_j = j - 30 * rate;
                if(decreased_j < 0) // 負になってしまった場合画素が存在しないの
                // で0にする
                    decreased_j = 0;
                dst_img.at<Vec4b>(Point(i, j)) =
                    background_img.at<Vec4b>(Point(i, decreased_j));
            }
        }
    }

    // 画像の表示
    imshow("dst_window", dst_img);
    imshow("depth_window", depth_img);
    imshow("background_window", background_img);
    imshow("adjust_window", adjust_img);

    int key = waitKey(1);
    if(key == 'q'){
        break;
    }
    else if(key == 'b'){
        // 背景の取得 カメラ画像の大きさをdepthサイズにしたものを記録
        // する
        resize(camera_img, background_img, depth_img.size());
    }
}

return 0;
}
```

プログラム1 (続き) 光学迷彩

図1 入力画像

図3 背景

図2 デプスキー

図4 光学迷彩

すぎるとデプスが取れないので、結果的にKinectに近付くと迷彩が解除され、図5のような光学迷彩らしい効果を得ることができます。背景差分などでデプスを用いない光学迷彩を作った場合、図5のように、対象までの距離を考慮した処理を行うことはできません。

光学迷彩は、① デプス情報を用いてRGB画像を処理する、② ピクセル単位の処理を行う、③ インタラクティブである、などの理由からサンプルに適切であると考えます。ソースコードについて解説を行います。

```
#include "kinectYnuiYKinect.h"
#include "kinectYnuiYImageFrame.h"
using namespace kinect::nui;

#include "kinect_utility.h"
```

ここでは、中村薫氏のラッパーのヘッダファイルをインクルードしています。また後に述べるデプス調整のために、われわれTA側で用意したutilityファイルもインクルードします。

```
// kinectクラスの宣言と初期化
Kinect kinect;
kinect.Initialize(NUI_INITIALIZE_FLAG_USES_COLOR | NUI_INITIALIZE_FLAG_USES_DEPTH_AND_PLAYER_INDEX);
```

```
// ストリーム作る。画像用とデプス用
ImageStream& video = kinect.VideoStream();
video.Open(NUI_IMAGE_TYPE_COLOR, NUI_IMAGE_RESOLUTION_640x480);
```

```
ImageStream& depth = kinect.DepthStream();
depth.Open(NUI_IMAGE_TYPE_DEPTH_AND_PLAYER_INDEX, NUI_IMAGE_RESOLUTION_320x240);
```

ラッパーにて定義されるKinectクラスを用いてデータにアクセスします。このように、生ポインタを介せずにRGB画像もデプスも同様の形式で取得することができます。SDK betaではデプスの解像度はRGB画像の半分になります。

図5 光学迷彩らしい効果

図6 すれ補正前

```
//カメラ画像をデプス画像に微調整するためのエンジン
ColorFromDepthEngine CFDengine;
```

utilityにて定義したクラスを用いて、RGB画像とデプスの微調整を行います。このクラスによる処理を毎フレーム介すことで、RGB画像とデプスのずれがなくなり、デプスをキーとしてRGB画像に処理を行うことができます。これにより、図6のようにずれていたものが図7のように補正されます。

```
// データの更新を待つ
kinect.WaitAndUpdateAll();

// 次のフレームのデータを取得する
ImageFrame image( video ); // カメラから画像
DepthFrame depthMD( depth ); // デプスカメラから
デプス画像

// データのコピー
camera_img = Mat(camera_img.size(), CV_8UC4,
(BYTE *)image.Bits());
depth_img = Mat(depth_img.size(), CV_16UC1,
(BYTE *)depthMD.Bits());

// depthに合わせて微調整した画像の取得（サイズは
デプスに合うように小さくなっている）
adjust_img = CFDengine.getAdjustedImage
(camera_img);
```

ここからは処理本体のwhileループになります。まずは毎フレーム、RGB画像とデプスの準備をします。waitAndUpdateAllはデバイスが画像を取得するまで自動的に待ってくれるので、明示的に何ms待機するなど記述する必要はなく簡単です。一度ImageFrame、DepthFrameという構造を介し画像を取得し、OpenCVのMat形式に変換します。またここでRGB画像・デプスの微調整も行います。これにてRGB画像・デプスの準備が整いました。

図7 すれ補正後

```
// 以下が光学迷彩のアルゴリズム
dst_img = adjust_img.clone(); // 微調整済みカメラ画像
// 結果表示画像にコピー
for(int j=0; j<depth_img.rows; j++){
    for(int i=0; i<depth_img.cols; i++){
        short depth_value =
depth_img.at<ushort>(Point(i, j)); // 注目点のdepthの
値を記録
        if(depth_value != 0){
            double rate = (DEPTH_MAX - depth_value) /
(DEPTH_MAX - DEPTH_MIN); // 近いほど値大きい
(0から1)
            int decreased_j = j - 30 * rate;
            if(decreased_j < 0) // 負になってしまった場合画
素が存在しないので0にする
                decreased_j = 0;
            dst_img.at<Vec4b>(Point(i, j)) =
background_img.at<Vec4b>(Point(i, decreased_j));
        }
    }
}
```

光学迷彩本体のアルゴリズムになります。RGB画像・デプスは大きさがそろっており、その各ピクセルに対し逐次処理を行います。ここでは簡便さのため速度を考慮せず、atメソッドで画素アクセスを行います。RGB画像の各ピクセルに対し、対応するデプスの値をdepth_valueとします。ここでdepth_valueが0でない、すなわち、Kinectがデプスを取得できている（遠すぎるまたは近すぎてKinectがデプスを取得できない場合0になります）とき、その画素に対し処理を行うdepth_valueを0～1の範囲に正規化（rate）し、その値に応じて背景画像の対応点から上にオフセットをとり、背景画素を抜き取り、それを重畳することで透明感を表現します。すなわち、デプスを検出している領域は、デプスの値に応じて背景の少し上の部分の画像をコピーします。これにより、完全に位置が合っているよりも少しずれて透明人間らしさが演出されます。

3.4 講義四日目：自由課題

四日目は自由時間としました。光学迷彩の課題を引き続き行う、また最終課題について考える、などを行いました。

4. 考察・感想

以上が前半の4回の講義になります。Kinectのように自分の動きを直接取得できることはとても直感的であり、受講者は皆楽しんでプログラミングを行っていました。Kinectを使った講義という、ある程度プログラミング経験がある学生を対象として、何か作ってこさせるものが多いと思いますが、逆に、プログラミング初学者相手の講義でも、その直感性は楽しさに直結するので面白いと思います。

しかし、OpenCVなどによりハードルは下がっているとしても、初学者にとって画像処理プログラミングは必ずし

も簡単ではないので、環境構築やサンプルプログラムはよく考えるべきだと思います。

本講義ではKinectのトラッキングの原理論文など紹介に留めましたが、現在Kinect関連技術は、原理的な面から応用的な面まで広く研究が進んでいます。論文をもっと紹介するのも良いと思います。実際に自分が動かしたものの理論を知ること、今後の研究への道を示すかもしれません。

5. 参 考 資 料

書籍として入手できるものはすぐに情報が古くなってしまっているのが現状です。現在最新のものとしては、C++対応の文献2)があります。また文献3)ではロボット連携の話など面白い応用について言及されています。また、文献4)は薄い本ですが、TOFの原理やデプスの超解像など技術的なトピックを扱っています。最新の情報に触れるには、文献1)を始め、ネット上のリソースにあたると良いと思います。

（2012年6月1日受付）

〔文 献〕

- 1) かおるんダイアリー, <http://d.hatena.ne.jp/kaorun55/20110621>
- 2) 中村薫, 齋藤俊太, 宮城英人: "KINECT for Windows SDKプログラミングC++編", 秀和システム (2012)
- 3) G. Borenstein: "Making Things See: 3D vision with Kinect, Processing, Arduino and MakerBot", O'reilly (2012)
- 4) C.D. Mutto, P. Zanuttigh, G.M. Cortelazzo: "Time-of-Flight Cameras and Microsoft Kinect™", Springer (2012)

まつい ゆうすけ 2011年、東京大学工学部電子情報工学科卒業。現在、同大学大学院学際情報学府修士課程在学中。コンピュータビジョン、画像処理、特に漫画・線画に対する研究に従事。