平成 26 年度　特別研究報告書

衣類群からの衣類分離のための 3D 情報解析の検討

龍谷大学　理工学部　情報メディア学科

T110404　伊藤　舞貴

指導教員　三好　力　教授

内容梗概

現在，日常生活をサポートするために多種多様なサービスが提供されている．家事の自動化もその一つである．しかし家事の自動化は，家事全般を網羅するわけではなく人の操作が必要な場合も多い．また自動化には処理空間の現在の状況を取得して自己判断を行う必要もある．例えば，全自動掃除機ルンバは搭載されたセンサーを利用し，作業する空間の情報を取得して行動を実行する．本研究では，洗濯物処理作業の過程において自動化が確立されていない部分である衣類群の自動分別に着目し，情報を取得するセンサーとして Kinect センサーを用いて，乾燥した複数の衣類群から単一の衣類の分類を行う衣類判別技術の確立を目指す．

目次

第 1 章 はじめに

1.1 研究背景

現在，日本の世帯数はおよそ 5500 万世帯であり，その内共働きの世帯は約 1260 万世帯存在する[1]．一人暮らしの世帯は約 1680 万世帯であり，その中でも高齢者の世帯は 480 万世帯存在する[1]．それらの人々の中には家事に割く時間を捻出することが困難な場合や家事に労力を使うことが困難だと考える人も多くいる．この悩みを解消するために様々な発明やサービスが存在する．食事に使用した食器や調理器具は全自動食洗機により手軽な操作で綺麗にされる(図 1)．全自動掃除機ルンバは，予め設定しておくことで定期的に部屋を掃除する(図 2)．全自動洗濯機に洗濯物と洗剤を投入するとボタンを数回操作するだけで手軽に汚れは落ちる．夕食の食材を届けてくれるサービスや高齢者向けの買い物代行サービスも存在する．当人たちに取って代わり，日常生活をサポートすべく現代社会では様々なサービスが実現している．

図 1.1.1 全自動食洗機

図 1.1.2 全自動掃除機ルンバ

しかし家事を行う一連の流れを網羅するわけではなく，人の操作による特定の条件を満たすことが必要な場合も多い．前記の例に対する条件を挙げるならば，収拾したゴミを廃棄する部分や食器を装置に固定することが特定の条件を満たすために必要な人の操作に該当する．その中でも衣類を洗濯する過程における乾いた衣類群が乱雑に積み重なる状況において分別と収納する作業は，未だ自動化が一般化していない部分に該当する．自動化は進んでいるが，すべての工程が機械による自動化がなされているわけではないのが現状である．

全自動掃除機ルンバは搭載されたセンサーを利用し，作業する空間の情報を取得して行動を実行する．同様に，衣類群の自動分別には作業する空間の状況を把

握して行動を選択する必要がある．本研究では情報を取得するセンサーとしてKinect センサーに着目する．Kinect センサーは空間の情報を取得することに優れるRGB センサーと距離センサーを持つ．これらセンサーから取得した情報を用いて衣類の判別を行う．

本研究では洗濯物処理作業の過程において自動化およびサービスが確立されていない部分である，複数の衣類が混在する衣類群から一つを取得するシステムの開発を目的とする．

第 2 章 既存技術

2.1 概要

Kinect では搭載された多種のセンサーやモーションキャプチャなどの特徴を利用して，娯楽や教育，福祉など幅広い分野において活用されている．画像処理の分野においても利用されており，衣服を特定する技術に関しても研究・開発が進められている．本章では，上記の既存技術についての説明を以下に示す.

2.2 深度情報を利用した任意の距離の抽出

Kinect に搭載されているセンサーの一つである深度センサーから取得できる情報を利用した物体検出方法．Kinect に搭載されている深度センサーはおよそ 850～4000 の範囲で深度の検出を行うことが可能である(単位はミリメートル)．任意の範囲を指定してその深度に該当するピクセルのみを描画することで，特定範囲の抽出が可能となる．[2]

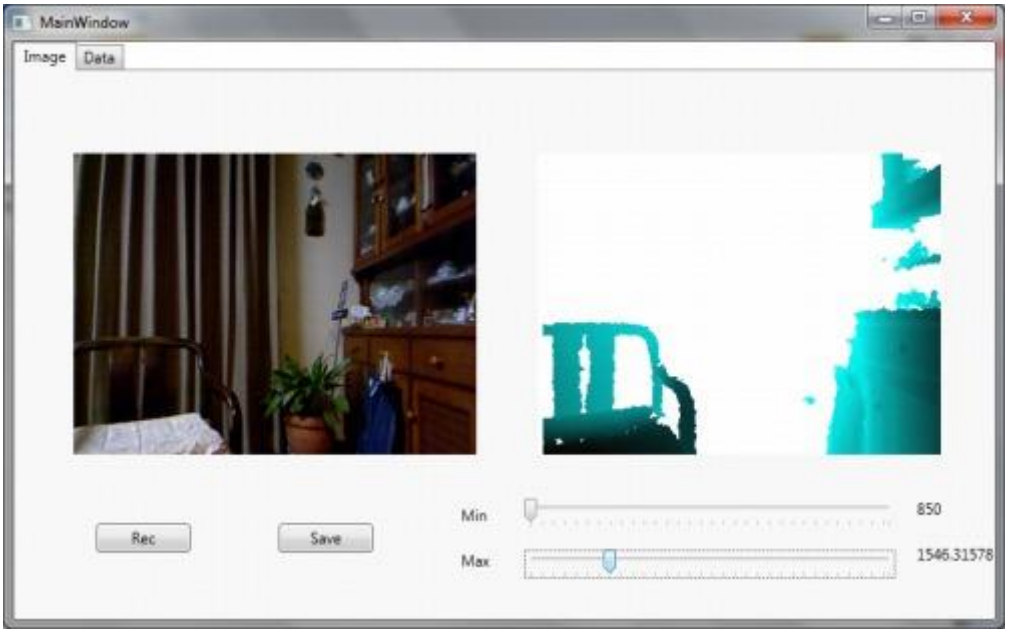

図 2.2.1 任意の距離の抽出

2.3 最近傍法による物体認識

最近傍法は，前もって格納されているデータと入力されたデータの比較を行い，最も近似しているデータを選択し，判断する手法．図 3.3.1 では画像データで入力し，格納されていたやかんのデータに近似していると判断している．[3]

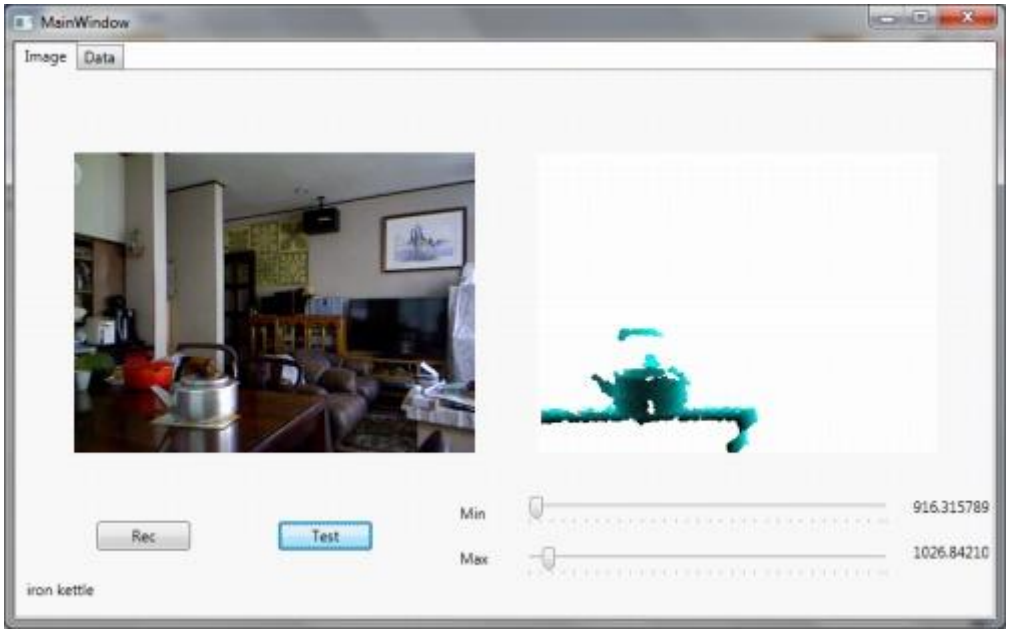

図 2.3.1 最近傍法による物体認識

2.4 複数センサを用いた画面領域分割

Kinect に搭載されている近赤外線距離画像センサを用いてフィルタを作成し画像の切り出しを行うことによってより複雑に物体が配置された画像から中心に配置された物体の領域を容易に抽出できるシステム．Kinect の近赤外線センサーから得られる深度画像および距離データを利用して，深度画像の 2 値化を行う(図 2.4.1)．この 2 値化したものをフィルタとして用いることで，画像領域分割を行う(図 2.4.2)．[4]

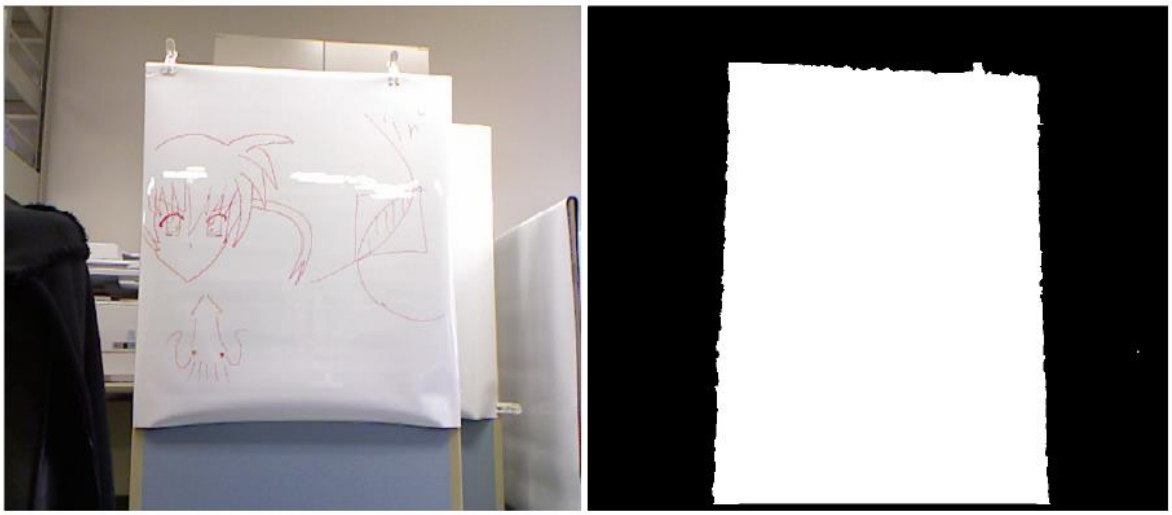

図 2.4.1 取得したカメラ画像(左)および作成したフィルタ画像(右)

図 2.4.2 マスク処理によって切り出された画像

2.5 3D 情報を用いた洗濯物片付き度合い判定システム

洗濯物が片付いている状態を図 2.5 のように複数の段階に分けて定義し，ゲームコントローラ Kinect によるカメラ画像と 3 次元情報を用いて定義とのマッチングを行うことにより洗濯物の片付け度合い判定システム． Kinect センサーを用いて，監視範囲領域内に存在する物体の形状や位置等の条件の組み合わせで現在の段階の判断を行う．[5]

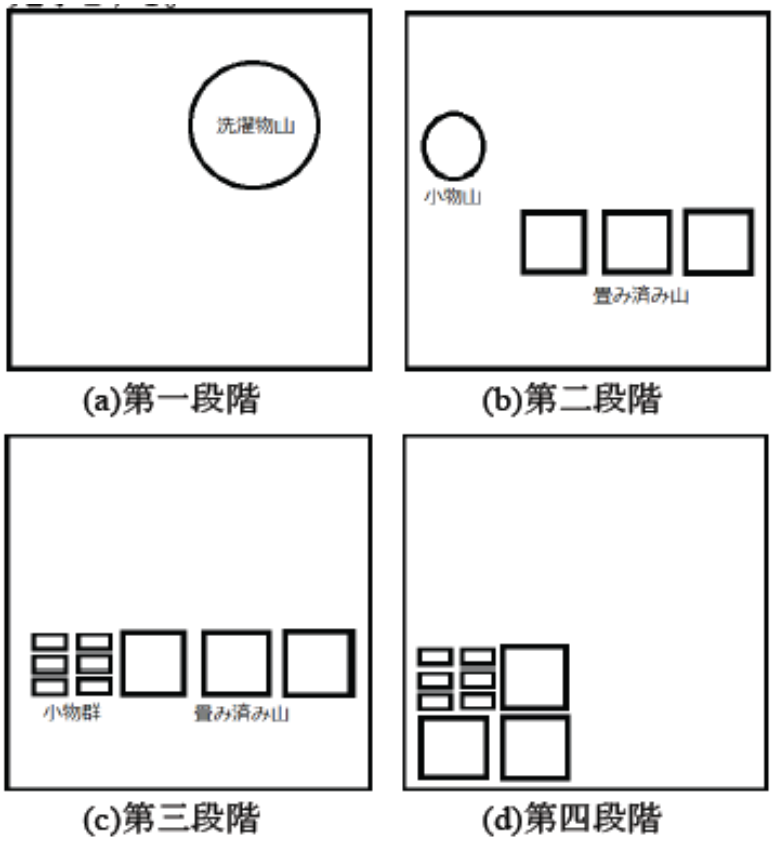

図 2.5 洗濯物片付けの段階分け

第 3 章 提案手法

3.1 概要

洗濯物に関する生活支援装置として全自動洗濯機が存在するが，衣類の処理作業すべてを行うわけではなく収納までは行わない．人の手を介さず自動で衣類の整理を行うシステムを考えると，機器のみで衣類の状態の判別や作業を行えることが望ましい．この場合，現在人の手によって処理されている洗濯物片づけ作業を自動で行えるロボットもまた求められるロボットの一つということになる．ロボットが洗濯物片づけ作業を行う場合，洗濯物が混在する中から一つのみを選択し，取得する処理が必要である．

本研究では，複数のセンサーが搭載されている Kinect を用いて，Kinect センサーから入力される RGB 情報および深度情報を使用し，複数の衣類が混在する衣類群から一つを取得するためのクリッピング範囲を決定するシステムの開発を行う．複数の物体を監視している Kinect センサーから RGB 情報および深度情報を 640x480 の画像として取得を行い，処理に利用している[6]．処理のフローチャートを図 3.1 に示す．

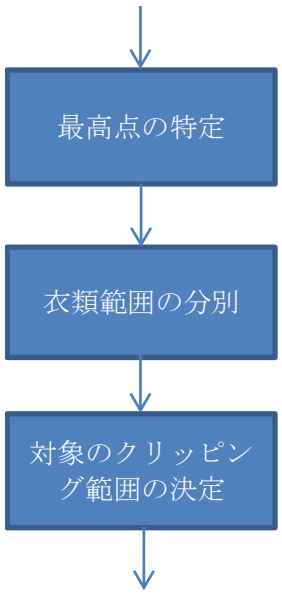

図 3.1 処理のフロー

3.2 最高点の特定

この処理では，複数の衣類が重なった状態で存在するとき，できる限り簡単に一つの衣類を取得するために Kinect から得た情報から混在する衣類の最も上部に

存在する衣類を特定する事を考える．最も上部に存在する衣類の特定には入力された深度情報を用いる．対象群上方に設置された Kinect センサーから対象物となる衣類までの距離を深度情報より取得し，Kinect センサーから最も近距離に存在するポイントを特定する[7]．Kinect センサーが上方に設置されていることより，Kinect センサーから最も近い場所が空間の最も高い位置であることが推定できる．これを利用して空間で最も高い位置に存在するポイントの特定を行う．処理のフローを図 3.2 に示す．

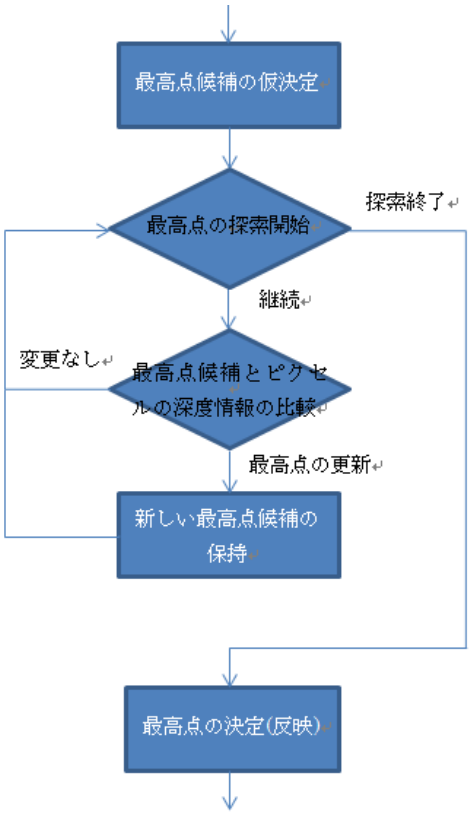

図 3.2 処理のフロー

3.3 衣類範囲の分別

この処理では，Kinect センサーから取得した RGB 情報を用いてグレースケール化を行い，輪郭抽出・エッジ検出を用いて 2 値化画像を作成する．これにより，対象の輪郭による各物体の同色範囲を線引き・分割が可能となる．グレースケール化には OpenCV ライブラリの関数 cvtColor を用いる．グレースケール画像から 2 値化画像に変換する際はノイズの混入する程度を考慮して，閾値は複数の候補を試行して最適なパラメータを求めた[8]． 処理のフローを図 3.3 に示す．

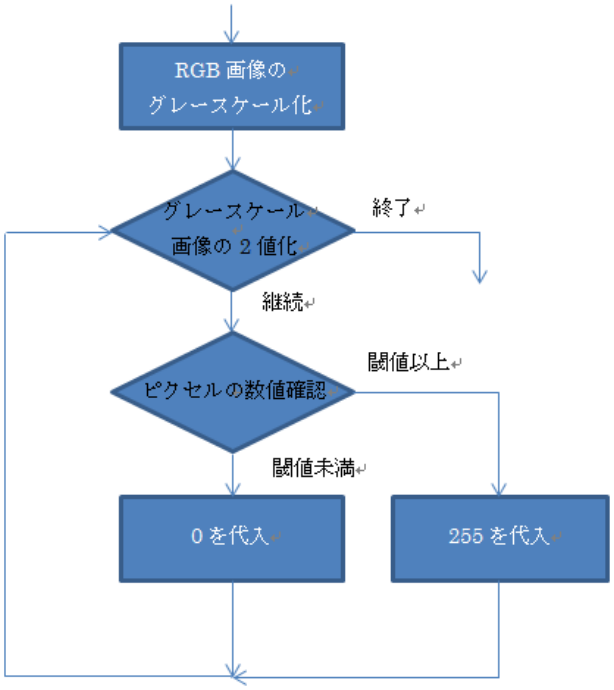

図 3.3 処理のフロー

3.4 対象のクリッピング範囲の決定

この処理では，これまでの工程で Kinect センサーから取得した深度情報と RGB 情報により，空間の最も高い位置に存在するポイントの特定および対象の輪郭による各物体の範囲を線引き・分割を行った．本工程ではこれらの特定した情報を利用して空間の最も高い位置に存在するポイントを含む物体の範囲の推定を行う．

RGB 情報より作成した 2 値化画像上に深度情報より特定したポイントを重ね合わせ，画像上において空間の最も高い位置に存在するポイントから水平方向と垂直方向に輪郭までの直線方向の探索を行う．2 値化画像で輪郭が発見されるまで探索は行われる．これにより，画像上において空間の最も高い位置に存在するポイントから探索された十字部分を対象のクリッピング範囲と決定する．決定したクリッピング範囲を掴むことで衣類群から対象物を取得することが可能である．

2 値化画像において空間の最も高い位置に存在するポイントから垂直方向及び水平方向に探索を開始し，輪郭部分を発見するまで探索を行う．これにより，2 値化画像において空間の最も高い位置に存在するポイントから探索された部分が対象のクリッピング範囲と決定することが可能となる．処理のフローを図 3.4 示す．

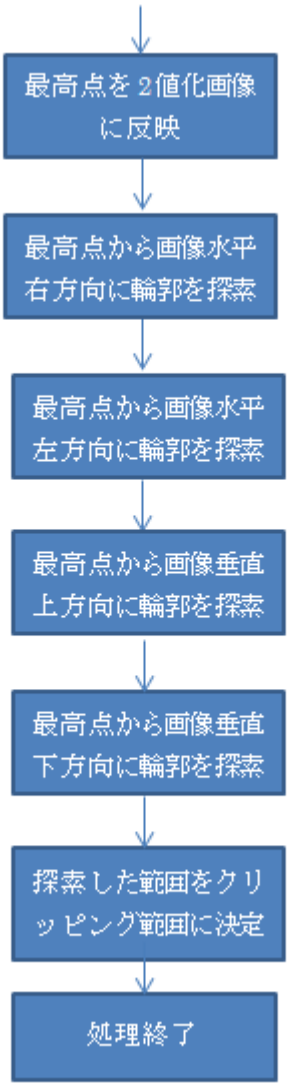
最高点を2値化画像に反映
最高点から画像水平右方向に輪郭を探索
最高点から画像水平左方向に輪郭を探索
最高点から画像垂直上方向に輪郭を探索
最高点から画像垂直下方向に輪郭を探索
探索した範囲をクリッピング範囲に決定
処理終了

図 3.4 処理のフロー

第 4 章 実験と考察

4.1 実験概要

本章では，Kinect センサーから取得可能な RGB 情報と深度情報の二つの情報を用いて処理を行うことで，センサーの範囲内に存在する最も上部に存在する物体を取得するためのクリッピング範囲の決定を試みた．

提案手法において，Kinect センサーから取得した RGB 情報および深度情報を用いて，2 値化画像を作成し，特定の深度情報と合わせることにより範囲を推定することができるか，処理の評価と考察を行った．評価は各項に基準を設け，主観による判断を行った．

4.2 実験方法

今回行った実験の処理を以下に示す．

- 空間の最も高い位置に存在するポイントの特定
- グレースケール化および輪郭抽出による 2 値化画像の作成
- 空間の最も高い位置に存在するポイントを含む十字範囲の探索

4.3 実験環境

本実験では Kinect センサーの認識距離を 1.0 メートルから 2.0 メートルに設定し，対象物の上方 1.5 メートルの位置に設置して認識を行った．開発環境は Visual C++ 2010，センサーから取得した情報の処理に OpenCV 2.2.0 を用いた．

4.4 実験結果と考察

4.4.1 空間の最も高い位置に存在するポイントの特定

提案手法 3.2 の手法で最高点の特定を行い，図 4.2 では，円の部分が空間の最も高い位置であると示されている．この処理では，数十回に一回ほど最高点ではない箇所が誤って検出される場合があった．近赤外線センサーの誤動作だと推測できるが，無視できる程度の頻度であると考えた．

表 4.1 に実験記録を示す．記録取得の際には 50 回の試行のうち 49 回の特定に成功した．

表 4.1 最高点特定の結果

試行回数	特定成功数	成功率
50	49	98%

図 4.2 ポイントを示す RGB 画像

4.4.2 グレースケール化および二値化画像の作成

提案手法 3.3 を用いて，グレースケール化および 2 値化画像の作成を行う．図 4.3 が RGB 情報よりグレースケール化を行った画像である．[8]

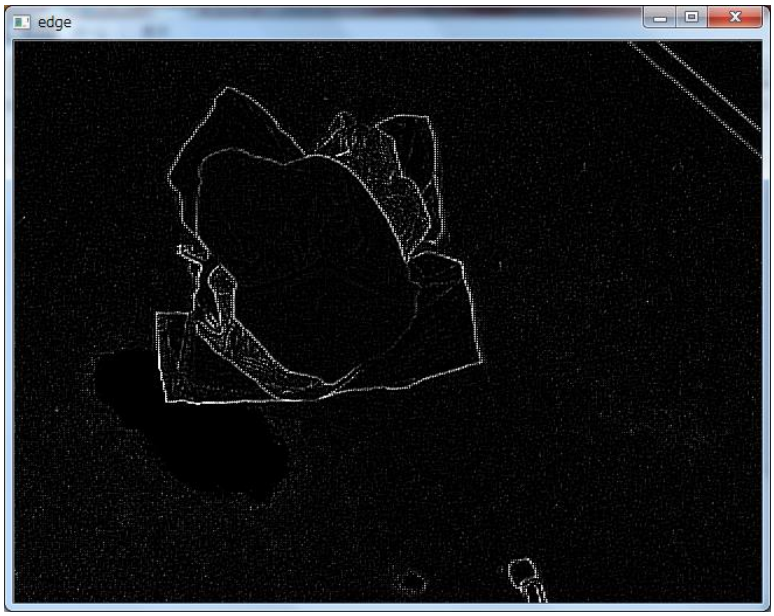

図 4.3 グレースケール化画像

図 4.3 よりグレースケール化処理を行ったことで，比較的はっきりと見てとれる程度に輪郭を抽出することに成功している．

探索処理を簡単にするために，さらにこのグレースケール化画像を 2 値化処理した画像が図 4.4 となる．

図 4.4 2 値化画像

視覚的に大きな変化は見られないが，画像全体の白色部分がはっきりと変化していることがわかる．図 4.3 および図 4.4 より，本工程での目的である対象の輪郭抽出に成功していると判断できる．今回の実験では閾値に 85 を用いた．

この処理では，図 3.2 の RGB 画像と図 4.3 のグレースケール化画像および図 4.4 の 2 値化画像を比較すると，黒い衣類とダークグレーの衣類の境界のように輪郭が適切に明確化されていない部分が存在することがわかる．複数回の試行からは前述のような欠落は少なくない結果が出ている．しかし輪郭部分すべてが欠損するような事例は稀であった．

表 4.2 および表 4.3 に実験記録を示す．グレースケール化の項目は，以降の処理に影響が出るか否かを基準として全ての輪郭が抽出されているかを確認して成否の決定を行った．2 値化画像作成の項目は，作成した際に輪郭が明確になったか否かを基準として成否の決定を行った．各項目における輪郭全損数とは，最高点を含む対象の輪郭が部分的欠損ではなくグレースケール化画像または 2 値化画像にお

いて認識不可能な規模の欠損であった場合である．表 4.2 および表 4.3 から 2 値化画像処理を行う際に閾値によって輪郭が欠損する場合が多い結果となっていることがわかる．

表 4.2 グレースケール化の結果

試行回数	特定成功数	成功率	輪郭全損数
50	34	68%	3

表 4.3 2 値化画像作成の結果

試行回数	特定成功数	成功率	輪郭全損数
50	24	48%	12

4.4.3 空間の最も高い位置に存在するポイントを含む範囲の探索

提案手法 3.4 を用いて，これまでの工程で行ってきた処理で獲得した，空間の最も高い位置に存在するポイントおよび 2 値化画像を利用して図 4.5 のように探索を行い，この処理結果をわかりやすいように RGB 画像上に重ね合わせて加工したものが図 4.6 となる．

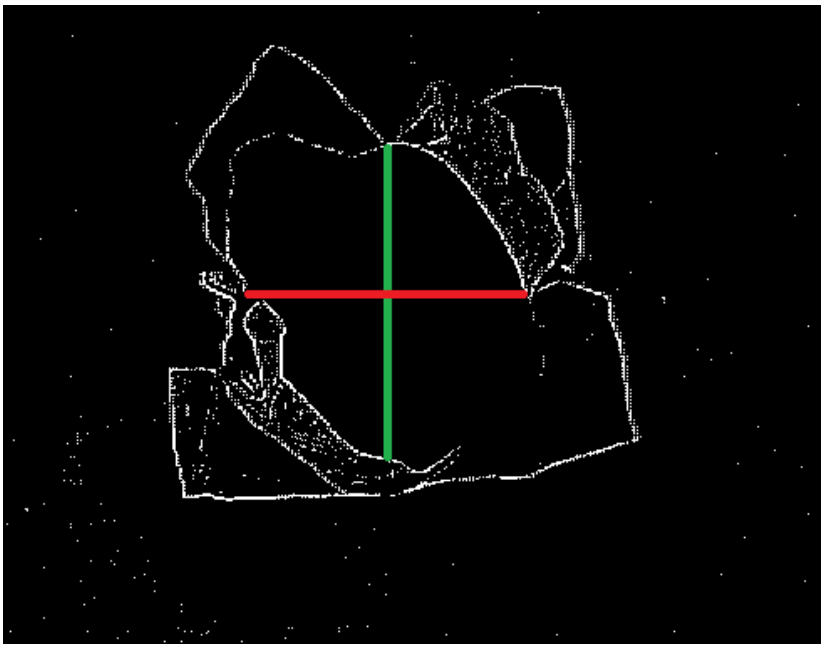

図 4.5 加工した 2 値化画像

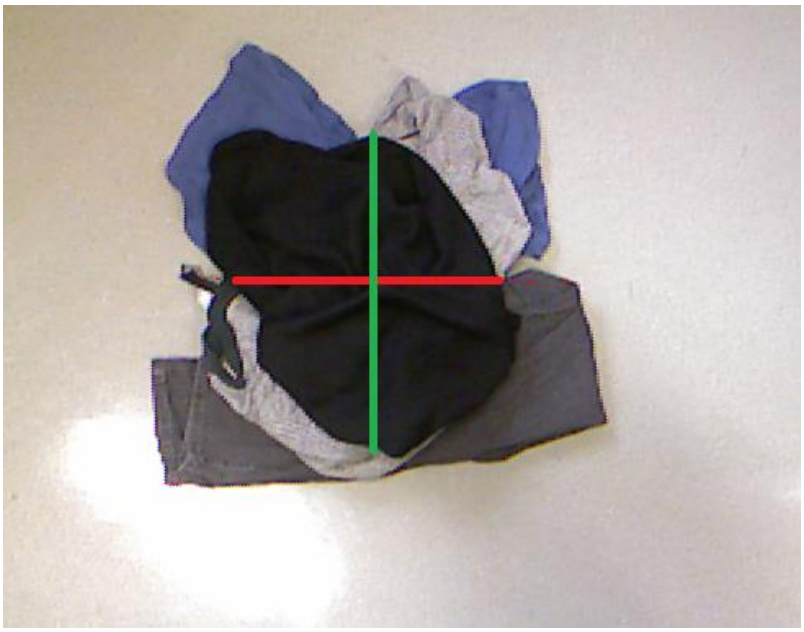

図 4.6 加工した RGB 画像

図 4.5 において，2 線の交点に空間の最も高い位置に存在するポイントが存在し，輪郭まで 2 線を延長した部分は空間の最も高い位置に存在するポイントを含む範囲と断定することが可能であり，空間の最も高い位置に存在するポイントを含む垂直方向および水平方向範囲の推定ができた．複数回の試行の結果，前行程の欠損が少なくない頻度で発生する影響で，本工程の処理にも影響を受ける事例が存在した．具体的な事例としては，2 値化画像の輪郭欠損の影響で，対象物の実際の輪郭が正常に抽出できず，探索が適正な位置で終了しない場合が存在した．しかし部分欠損の事例は少なくないが，全てが欠損している事例は無視できる程度に稀であったため，4 方向の候補の内残りが使用できる状態であればその範囲を使用することが可能であるため，クリッピング範囲を推定する目的は達成されていた．

実験記録を表 4.4 に示す．全行程の結果は，最高点の特定・グレースケール化・2 値化画像の作成，全ての工程を成功した場合の成功率が算出されている．前述した通り輪郭の欠損が一部の場合許容できるため，許容した場合を含む結果を表 4.5 に示す．

表 4.4 全行程の結果

回数	特定成功数	成功率
50	24	48%

表 4.5 欠損を許容した全行程の結果

回数	特定成功数	成功率
50	35	70%

4.5 考察

空間の最も高い位置に存在するポイントは概ね問題なく特定することができた．グレースケール化および 2 値化画像の作成は対象によってノイズが発生したが，処理に大きく問題がでない程度でない程度であり，最高位に存在する衣類を取り出すための最高点を含む取得可能範囲を決定することができた．対象物を取得可能な範囲を掴むことで，人間の手によって衣類群から対象のみを取得することができた．

輪郭抽出段階において，部分的に輪郭が欠落した事例が複数存在した．複数回の試行から主な原因は，対象物の模様や光源の位置による陰影の影響が大きいと推測できた．

2 値化画像の作成工程において，前項まで用いていた例でも確認できるが，グレースケール化画像で存在した輪郭線が消えている部分が存在する．これは 2 値化画像作成時の閾値の設定に関係することがわかるが，一定以上の輪郭線が抽出できなかった場合に 2 値化処理の段階で輪郭線が部分的に欠落してしまう現象である．ノイズ除去を考慮に入れた場合，一定の欠落の存在は認めざるをえないと考えられるが，表 4.3 からもわかるように閾値による欠損は改善の余地があると考えられた．

第 5 章 おわりに

本論文では 3D センサーが搭載されている Kinect センサーによる RGB 情報および深度情報を用いた衣類群の解析を行い，対象物を取得するためのクリッピング範囲を特定することができた．近赤外線センサーのノイズからか深度情報の取得が安定しない場合が存在し，これにともない最も高所にあるポイントの特定が安定しない場合が散見された．また輪郭抽出においても抽出が安定しない種類の対象物が存在することが確認できた．しかし部分的に不安定な場合であれば対象物を取得するためのクリッピング範囲を決定することは可能であった．

今後の課題として，今回 Kinect センサーから取得できる情報に多くのノイズが含まれていた可能性が存在することが推測でき，近赤外線センサーの精度によるものだと考えられるが，これによる影響は誤差として排除できるほどの大きさではない場合がある．そのためプログラム面でのノイズの抑制または除去を行う必要がある．また，2 値化画像を作成する際の閾値を環境によって変更することで探索の成功率上昇が推測できる．探索処理をより多方向に多重に繰り返すことでより広範囲を取得するための候補として特定することが可能であると推測できるが，処理が多く複雑になるため現実的ではないと考える．

謝辞

本論文を作成するにあたり，ご指導頂きました三好力教授に深謝いたします．また，多忙の中日頃の議論にご協力くださった同三好研究室の皆様や，学友の皆様に心より感謝いたします．

参考文献

[1]. 総務省
<http://www.soumu.go.jp/>

[2]. Kinect センサーから任意の範囲の深度マップを抽出
<http://d.hatena.ne.jp/astrobot/20110701/1309528359>

[3]. Kinect と最近傍法で物体認識
<http://d.hatena.ne.jp/astrobot/20110706/1309925301>

[4]. 広瀬大樹，三好力，“複数センサを用いた画面領域分割”，情報処理学会第 75 回全国大会

[5]. 広瀬大樹，三好力，“3D 情報を用いた洗濯物片付き度合い判定システム”，情報処理学会第 76 回全国大会

[6]. 画像処理 – OpenCV – CookBook
<http://opencv.jp/cookbook/opencv_img.html>

[7]. 中村薫，宮城英人，齋藤俊太，“KINECT for WindowsSDK プログラミング C++編”，秀和システム

[8]. OpenCV.jp
<http://opencv.jp/>

付録

```
#include <iostream>
#include <sstream>
#pragma warning(disable:4996)
//NuiApi.h の前に windows.h をインクルードする
#include <Windows.h>
#include <NuiApi.h>
#include <opencv2/opencv.hpp>

//debug mode
#pragma
comment(lib,"C:¥¥OpenCV2.2¥¥lib¥¥opencv_core220d.lib")
#pragma
comment(lib,"C:¥¥OpenCV2.2¥¥lib¥¥opencv_imgproc220d.lib")
#pragma
comment(lib,"C:¥¥OpenCV2.2¥¥lib¥¥opencv_highgui220d.lib")
#pragma
comment(lib,"C:¥¥OpenCV2.2¥¥lib¥¥opencv_objdetect220d.lib")

#define ERROR_CHECK( ret )  ¥
if ( ret != S_OK ) {    ¥
  std::stringstream ss;¥
  ss << "failed " #ret " " << std::hex << ret << std::endl;¥
 throw std::runtime_error( ss.str().c_str() );  ¥
}
const NUI_IMAGE_RESOLUTION CAMERA_RESOLUTION = NUI_IMAGE_RESOLUTION_640x480;
class KinectSample{
private:
INuiSensor* kinect;
HANDLE imageStreamHandle;
HANDLE depthStreamHandle;
HANDLE streamEvent;
DWORD width;
DWORD height;

public:
KinectSample(){
}
~KinectSample(){//終了処理
 if( kinect != 0 ) {
kinect->NuiShutdown();
kinect->Release();
  }
}

void initialize(){
 createInstance();// kinect の設定を初期化する
ERROR_CHECK( kinect->NuiInitialize( NUI_INITIALIZE_FLAG_USES_COLOR|NUI_INITIALIZE_FLAG_USES_DEPTH ) );
// RGB カメラを初期化する
ERROR_CHECK( kinect->NuiImageStreamOpen( NUI_IMAGE_TYPE_COLOR,CAMERA_RESOLUTION,0,2,0, &imageStreamHandle ) );
// 距離カメラを初期化する
ERROR_CHECK( kinect->NuiImageStreamOpen( NUI_IMAGE_TYPE_DEPTH,CAMERA_RESOLUTION,0, 2, 0, &depthStreamHandle ) );
// フレーム更新イベントのハンドルを作成
streamEvent = ::CreateEvent( 0, TRUE, FALSE, 0 );
ERROR_CHECK( kinect->NuiSetFrameEndEvent( streamEvent, 0 ) );
// 指定した解像度の画面サイズを取得する
::NuiImageResolutionToSize(CAMERA_RESOLUTION, width, height );
}

void run()  {
```

```
cv::Mat image;
//メインループ
while(1){
// データ更新を待つ
 DWORD ret = ::WaitForSingleObject( streamEvent, INFINITE );
 ::ResetEvent( streamEvent );
 drawRgbImage( image );
 drawDepthImage( image );

// 画像を表示する
cv::imshow( "Kinect Sample", image );
// 終了のためのキー入力をチェック兼、表示のためのウェイト
//int key = cv::waitKey( 10 );
int key = cv::waitKey( INFINITE );
if( key == 'q' ) {
break;
}
if( key == 'n' ) {//n 入力で次のループへ
}
}
}
private:
void createInstance(){
// 接続されている kinect の数を取得する
 int count = 0;
 ERROR_CHECK( ::NuiGetSensorCount( &count ) );
  if ( count == 0) {
 throw std::runtime_error( "kinect を接続してください" );
// 最初の kinect のインスタンスを作成する
ERROR_CHECK(::NuiCreateSensorByIndex(0,&kinect ) );
// kinect の状態を取得する
HRESULT status = kinect->NuiStatus();
if(status != S_OK){
throw  std::runtime_error( "kinect が利用可能ではありません" );
        }
}

void drawRgbImage( cv::Mat& image )
{
// RGB カメラのフレームデータを取得する
NUI_IMAGE_FRAME imageFrame = { 0 };
ERROR_CHECK( kinect->NuiImageStreamGetNextFrame( imageStreamHandle,0, &imageFrame ) );
// 画像データを取得する
NUI_LOCKED_RECT colorData;
imageFrame.pFrameTexture->LockRect(0,&colorData, 0, 0 );
// 画像データをコピーする
Image=cv::Mat(height,width,CV_8UC4,colorData.pBits );
// フレームデータを解放する
ERROR_CHECK( kinect->NuiImageStreamReleaseFrame( imageStreamHandle, &imageFrame ) );
}

void drawDepthImage( cv::Mat& image ){
USHORT holdnear = 3800;
//保持よう変数            //---star1
int holdbox[256]={},box=0;
int count=0;
//cv::Mat gray(480,640,CV_8UC1); //---star2
cv::Mat gray;
cv::cvtColor(image, gray,CV_RGB2GRAY);    // グレスケ
cv::Mat tmp_img;
 cv::Mat laplacian_img;
 cv::Mat mid;
 cv::Laplacian(gray, tmp_img, CV_8UC1, 3); // ラプラシアンフィルタ
 cv::convertScaleAbs(tmp_img, mid, 1, 0);
```

```
cv::imshow("edge",mid);
cv::threshold(mid,laplacian_img,85,255,CV_THRESH
_BINARY);
cv::imshow("thresh",laplacian_img);
// 距離カメラのフレームデータを取得する
NUI_IMAGE_FRAME depthFrame = { 0 };
ERROR_CHECK( kinect->NuiImageStreamGetNextF
rame( depthStreamHandle, 0, &depthFrame ) );
//距離データを取得する
NUI_LOCKED_RECT depthData = { 0 };
depthFrame.pFrameTexture->LockRect(0,&depthDat
a, 0, 0 );
USHORT* depth = (USHORT*)depthData.pBits;
for ( unsigned int i = 0; i < (depthData.size /
sizeof(USHORT)); ++i ) {
USHORTistance= ::NuiDepthPixelToDepth( depth[i] );
if(i>305900) break;
 if(holdnear > distance && distance > 500){  // 最も近
いポイントを保存//---star1
holdnear = distance;
holdbox[box] = i;
box++;
  }
LONG depthX = i % width;
LONG depthY = i / width;
LONG colorX = depthX;
LONG colorY = depthY;

// 距離カメラの座標を、RGB カメラの座標に変換する
kinect->NuiImageGetColorPixelCoordinatesFromDep
thPixelAtResolution(
 CAMERA_RESOLUTION, CAMERA_RESOLUTION,
0, depthX , depthY, depth[i], &colorX, &colorY );
// 一定以上の距離を描画しない
/*int index = ((colorY * width) + colorX) * 4;//---star2
gryascale_1st
UCHAR* data = &image.data[index];
UCHAR* graydata = &gray.data[i];
graydata[0] =(UCHAR)(0.114*(double)data[0]
+ 0.587*(double)data[1] + 0.299*(double)data[2]
)*/
/*if ( distance >= 800) {//特定距離色変換
int index = ((colorY * width) + colorX) * 4;
UCHAR* data = &image.data[index];
data[0] = 255;
data[1] = 255;
data[2] = 255;
if(distance >= 1000){
data[0] = 0;
data[1] = 255;
data[2] = 0;
}
}
 if(distance > 800 && distance < 4000){//--star2 一つ
前と比較して distance 特定以上の差が存在する場合に色
づけ
 USHORT disx;
if(i == 0) disx = distance;
if(distance < disx-5 || distance > disx+5) {
int index = ((colorY * width) + colorX) * 4;
UCHAR* data = &image.data[index];
data[0] = 255;
data[1] = 0;
data[2] = 255;
disx = distance;
}
count++;
  }*/
}
cout << count << endl;
cout << box << endl;   //---star1
/*for(int j =holdbox[box-1]; j < holdbox[box-1]+100 ;
j++){        //---最近点を rgbWindows に可視化
LONG depthX = j % width;
LONG depthY = j / width;
```

```
LONG colorX = depthX;
LONG colorY = depthY;
// 距離カメラの座標を、RGB カメラの座標に変換する
kinect->NuiImageGetColorPixelCoordinatesFromDep
thPixelAtResolution(CAMERA_RESOLUTION,
CAMERA_RESOLUTION, 0, depthX , depthY,depth[j],
&colorX, &colorY );
int index = ((colorY * width) + colorX) * 4;
UCHAR* data = &image.data[index];
data[0] = 0;
data[1] = 0;
data[2] = 255;
}*/
int d=0;
for(int over = holdbox[box-1]；  over > width；over =
over - width){              //---最近点垂直上方に可視化
//int over = holdbox[box-1] - width;
 LONG depthX = over % width;
 LONG depthY = over / width;
 LONG colorX = depthX;
 LONG colorY = depthY;
 // 距離カメラの座標を、RGB カメラの座標に変換する
kinect->NuiImageGetColorPixelCoordinatesFromDep
thPixelAtResolution(CAMERA_RESOLUTION,
CAMERA_RESOLUTION, 0, depthX , depthY,
depth[over], &colorX, &colorY );
int index = ((colorY * width) + colorX) * 4;
UCHAR* edge = &laplacian_img.data[index/4];
if(edge[0] > 10){
UCHAR* data = &image.data[index];
data[0] = 0;
data[1] = 255;
data[2] = 0;
}
//if(d!=0 && edge[0] <10) break;
//if(over==holdbox[box-1]) cout << d << endl << over <<
endl;
//cout << d << endl << endl << endl;
d++;
}
int e=0;
for(int down = holdbox[box-1] ; down < (depthData.size
/ sizeof(USHORT)) ; down = down + width){   //--- 最 近
点直下方向に可視化
//int over = holdbox[box-1] - width;
 LONG depthX = down % width;
 LONG depthY = down / width;
 LONG colorX = depthX;
 LONG colorY = depthY;
 // 距離カメラの座標を、RGB カメラの座標に変換する
kinect->NuiImageGetColorPixelCoordinatesFromDep
thPixelAtResolution(
CAMERA_RESOLUTION,  CAMERA_RESOLUTION,
0, depthX , depthY, depth[down], &colorX, &colorY );
int index = ((colorY * width) + colorX) * 4;
UCHAR* edge = &laplacian_img.data[index/4];
/*if(edge[0] > 10){
UCHAR* data = &image.data[index];
UCHAR* dataC = &image.data[index+1];
if(10<e&&e<90){
data[0] = 0;
data[1] = 0;
data[2] = 255;
dataC[0] = 0;
dataC[1] = 0;
dataC[2] = 255;
}
else{
data[0] = 0;
data[1] = 255;
data[2] = 0;
dataC[0] = 0;
dataC[1] = 255;
dataC[2] = 0;
}
}*/
```

```
//if(e!=0 && edge[0] <10) break;
//if(over==holdbox[box-1]) cout << d << endl << over <<
endl;
//cout << d << endl << endl << endl;
e++;
}
int f=0;
int high = holdbox[box-1]/width;
for(int lr = high*width ; lr < (high+1)*width ;
lr++){//---最近点 suihei 方向に可視化
 LONG depthX = lr % width;
 LONG depthY = lr / width;
 LONG colorX = depthX;
 LONG colorY = depthY;
 // 距離カメラの座標を、RGB カメラの座標に変換する
kinect->NuiImageGetColorPixelCoordinatesFromDep
thPixelAtResolution(
 CAMERA_RESOLUTION, CAMERA_RESOLUTION,
0, depthX , depthY, depth[lr], &colorX, &colorY );
int index = ((colorY * width) + colorX) * 4;
UCHAR* edge = &laplacian_img.data[index/4];
if(edge[0] > 10){
UCHAR* data = &image.data[index];
//if(10<e&&e<90){
data[0] = 0;
data[1] = 0;
data[2] = 255;
/*}
else{
data[0] = 0;
data[1] = 0;
data[2] = 255;
}*/
}
f++;
}
cout << holdnear << endl << endl << endl;
// フレームデータを解放する
ERROR_CHECK( kinect->NuiImageStreamReleaseFr
ame( depthStreamHandle, &depthFrame ) );
}
};

void main(){
try {
KinectSample kinect;
kinect.initialize();
kinect.run();
}
catch ( std::exception& ex ) {
        std::cout << ex.what() << std::endl;
        }
}
```