平成 24 年度　特別研究報告書

センサフュージョンを用いた個人認証

龍谷大学　理工学部　情報メディア学科

T090458　米谷 和記

指導教員　三好 力　教授

内容梗概

Kinect とは Microsoft 社のゲーム機 Xbox360 で使用できるゲームコントローラの一種である．人物を認識し追跡が可能であること，複数のセンサを搭載していることなどから，看護や医療といった幅広い分野で開発・研究が行われている．その利用法の一つにホームネットワークシステムへの応用がある．ホームネットワークシステムとは家電同士をネットワーク接続することで家電の遠隔操作や同時操作を実現するものであるが，特定の人物に操作させないなどの操作制限は現在行うことができない．

そこで本研究では，Kinect の複数のセンサを用いることでより高精度な個人識別を行うシステムを実現する．複数のセンサから得た情報を相互に補完することで計測の精度を向上させる技術であるセンサフュージョン技術を用いることで Kinect を個人識別システムに利用することの有用性を確認する．

目次

第 1 章　はじめに

1.1　研究背景

近年，テレビゲーム機のゲームデバイスである Kinect が様々な方面で注目され始めている．Kinect とは Microsoft 社のゲーム機 Xbox360 で使用できるゲームコントローラの一種である．特徴として，人物を認識し追跡が可能であること，複数のセンサを搭載していること，USB インタフェースがあることが挙げられる．これによりゲームコントローラ以外の利用も多くなされており，高齢者や障害者の生活のサポートを行うシステムなど開発分野も幅広いものとなっている．その利用法の一つにホームネットワークシステムへの応用というものがある．

ホームネットワークシステムとは，家電をネットワークに接続しリンクさせることでより便利に使えるようにするシステムのことである．これにより，テレビやエアコンなどの家庭内にある家電製品やセンサ類を宅内外から制御や監視することができ，複数の家電製品の連携，省エネなどのサービスを実現できる．具体的には，テレビの電源を入れるだけで自動的にエアコンが作動し快適な空間を構築できる，ホームサーバにアクセスすることで外出時に鍵の施錠状況を確認するといったことが可能となる．

このシステムを利用する利点としては，上記のような日常生活をより便利・快適に過ごすためのサポートをしてくれる点である．欠点としては，導入コストが高額であること，個人の特定ができないなどがある．特に，小さな子どもに危険な家電を操作させないなどの制限をかける必要がある場合など，個人識別は大きな課題となっている．

実際に，Kinect をホームネットワークシステム利用のための家電リモコンとして使用した次世代住宅も存在しており，家電の操作をより直観的に行うことが可能となっている．しかし個人識別による使用制限機能のようなものはなく，利用可能な家電操作もまだ少ないのが現状である．

1.2　研究目的

現在の Kinect を用いた家電リモコンでは同時に複数の家電を制御することは可能であるが，特定の人物の使用を制限する機能は搭載されていない．この問題点を改善するためには個人識別を行えるようにすることが有効である． Kinect で個人の識別が可能になると，小さな子どもに家電の使用を制限することができたり，個人に適した環境を素早く構築することができるようになる．Kinect には複数のセンサが初めから搭載されているため，センサフュージョン技術を用いることで高い識別精度を持った個人識別システムを構築する．

第 2 章　Kinect とは

Kinect は Microsoft 社から販売されている家庭用据え置き型ゲーム機 Xbox360 向けのゲームデバイスである．2010 年 11 月 4 日に米国で発売され，同年 11 月 20 日に日本でも発売された．特徴として Kinect が内蔵するセンサによってプレイヤーの動きを捉え，コントローラを使用せずにゲームの操作を可能とすることが挙げられる．このような音声やジェスチャーなど「人の自然な動作」で入力を行うユーザインタフェースのことを Natural User Interface(NUI)という．Kinect には，3 つのセンサおよび専用ソフトウェアを動作させるプロセッサが内蔵されており，人間を検出しプレイヤーの姿勢や動きを認識することができる．これにより従来のようなボタン操作ではなく，プレイヤー自身の体を使って直観的にゲームを操作することが可能となった．内蔵されているセンサは，写真撮影や映像録画に用いられる RGB カラー映像認識用カメラセンサ，Kinect から対象物までの距離を測定するための近赤外線距離画像センサ，音声を認識するためのマイクセンサの 3 種類である．さらにユーザの動きに対応するため，上下の角度を自動調整する電動チルト機構も備えている．

Kinect は人物の動きをデジタル的に記録するモーションキャプチャ技術を使用しているが，上記のセンサを用いることで，キャプチャ時に着る特殊なスーツおよび検出時に使用するトラッカーが必要なくなっている．これによりカメラに被写体を映す事でプレイヤーから Kinect 本体までの距離を計測し，ユーザの骨格のさまざまな動きを検出，ゲーム内のキャラクターの動きにリアルタイムに反映させることが可能となっている．

図 1：Kinect 外観

対象物との距離を計測する近赤外線距離画像センサは，赤外線を照射する近赤外レーザとレーザ照射パターンを捉えるための近赤外カメラで構成されている．また，計測方法として三角測量を利用している．具体的にまず，Kinect に搭載した近赤外レーザから対象物にスポット光を複数，ランダムに配列したようなパターンを投影し，対象物に映ったパターンをカメラで捉える．そして，捉えたパターンのカメラからの見込み角を求める．これに，レーザから対象物にパターンを照射した角度と，レーザとカメラの距離を基にして，対象物までの距離を算出する．パターン内のスポット光を区別できれば，1 回の投影によってカメラがとらえた画面内の物体の複数点の距離計測が可能になる．

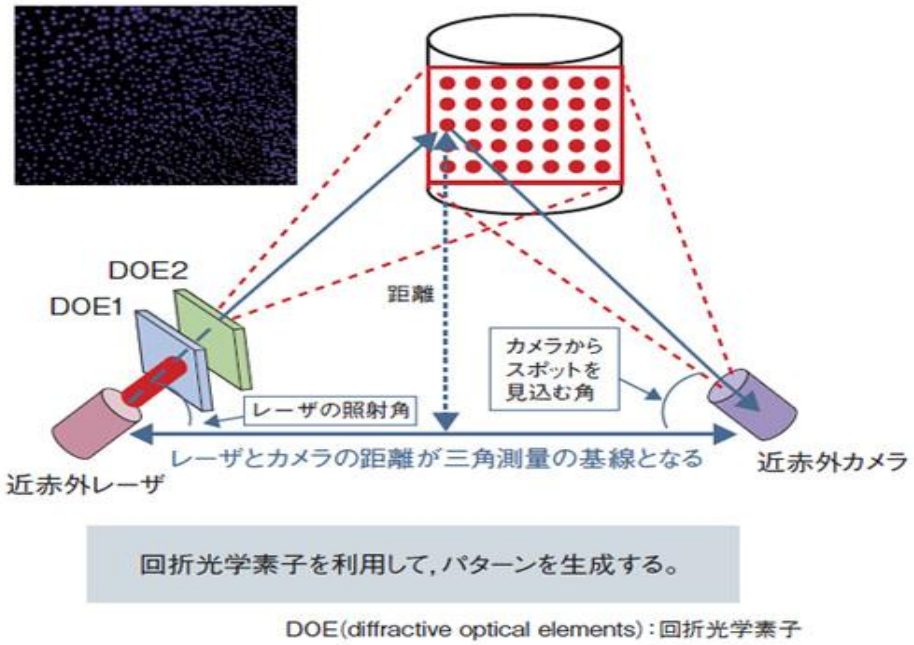

図2：三角測量を用いた距離測定

この距離画像センサから取得した距離画像を用いて，先に用意してある決定木により各部位のどこに相当するのかの識別を行っている．図3のように頭，肩正面，右肩，左肩，右肘，左肘，右手首，左手首，右手，左手，骨髄，骨盤正面，右骨盤，左骨盤，右膝，左膝，右足首，左足首，右足，左足の20箇所の動きを検出することができる．この20箇所をジョイントといい，これらの動きを追跡することで全身の動きを推定している．ジョイント各点はKinect本体から横方向をx，縦方向をy，奥行きをzとして3次元の値を所持している．

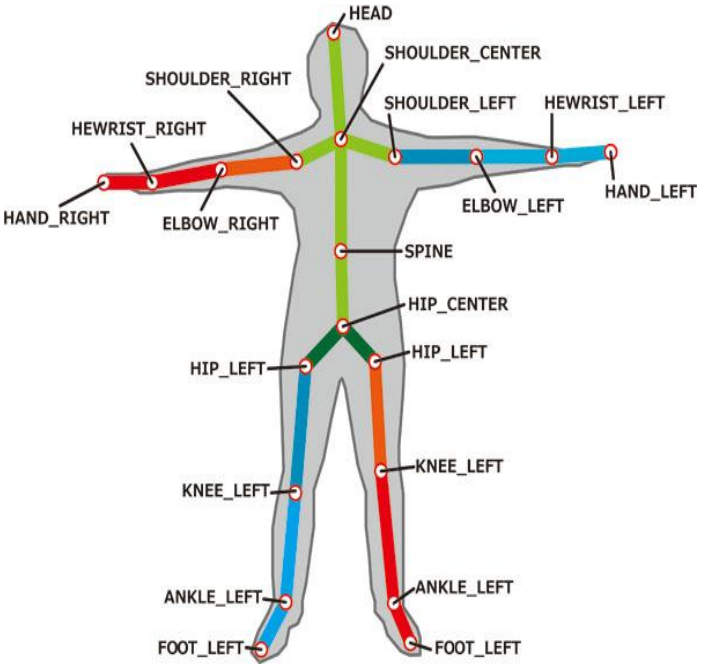

図3：ジョイントの検出箇所

また，Kinectはコンピュータに接続することも可能である．開発元のMicrosoft社は非商用の場合に限り「内部アルゴリズムにアクセスしないこと」と「Xbox向けのゲームの不正利用に使用しないこと」を条件としてシステム開発を許可している．これによりKinectは介護や医療など様々な分野で応用・研究されている．

第 3 章　既存技術

3.1　概要

Kinectは人物を認識し追跡が可能であること, 複数のセンサを搭載していることなどから，看護や医療といった幅広い分野で開発・研究が行われている．本章では，Kinectを用いた既存技術についての説明を以下に示す.

3.2　OAK（Observation and Access with Kinect）

OAK[2]とは, 脳性まひや脊髄性筋萎縮症などにより重度の障害を持つ人の口や手の動きをKinectで検出し，意思表示や能動的な活動を支援するソリューションである．従来では, わずかに動く指先や舌で物理的なスイッチを押すなどして意思表示を行っていたが，スイッチを常に身に着けなければならないなどの問題点があった．さらに，こうした活動支援には一般的に高価な機材が必要となり, 広く普及させることが困難であった．そこでOAKは，安価で手に入るKinectを用いることで従来の課題を解消することに成功している.

OAKの機能の例として，Kinectが顔の位置を認識すると，目（まぶた）の動きに応じてライトを操作できるフェイススイッチというものがある. 人によって任意に部位を動かせる範囲に差があるが, パソコン側で感度を調節することでユーザに最も適した設定が可能である.

図4：OAK使用の様子

3.3　wi-Go プロジェクト

wi-Goプロジェクト[3]は，スーパーなどにあるショッピングカートにKinectを取り付けることで，車椅子や妊婦といった体の不自由な人が買い物かごを持たずに店内を移動できるようにするシステムである．具体的には，Kinectが認識した人物の後をかごが乗ったカートが追従するといったものである．

図5：wi-Goプロジェクト使用例

3.4　側弯症計測システム

側弯症とは，脊柱が側方に湾曲したりねじれたりする病気である．痛みを伴うことは稀であり早期発見が難しいため，学校での検査が義務付けられているのだが，検査装置が高価，数値化による定量的な評価ができないなどの理由から検査がなかなか実施されないのが実情である．このシステム[4]はKinectで背中の形状を測定することで側弯症の進行度合いを数値として定量的に評価でき，病状の早期発見につなげるというものである．操作方法も簡単で，被験者にカメラ内に立ってもらい検査者が図6の矢印部分にあるボタンを押すだけで背中の高低差が保存される．

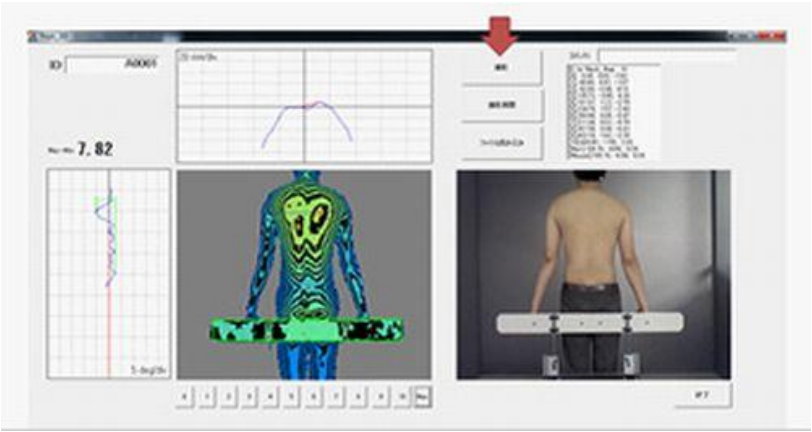

図6：システムの使用結果

3.5　家電操作システム

Kinectで家電を操作できる次世代住宅[5]が2012年の9月10日に公開された．Kinectを使った住宅設備の操作はリビングで行うことができる．現在利用が可能な操作は，TV・照明の操作，ブラインドの開閉の3種類である．利用者が操作に慣れるまでは混乱してしまう可能性を考慮し，できることを3つに絞り込んでいる．家電の操作は立ってだけでなく，椅子に座ったり寝転がったりした姿勢でも可能となっている．

図7：Kinectによるブラインドの開閉操作

3.6　個人識別システム

Kinectに搭載されている距離画像センサを用いた個人識別システム[6]である．個人識別を行うための特徴量は，距離画像センサから得られるジョイント間の距離を使っており長さを算出している部位は身長，右腿，左腿の3か所である．登録と照合の2つのモードを搭載しており，登録モードでは認識している人物が1人だけの場合に特徴量を算出し名前と共にデータベースに個人データとして記録される．照合モードではデータベースに記録されている特徴量と現在算出されている特徴量を比較することで個人を識別している．この際に設定されている見込み誤差は6～7%である．推奨されている使用距離はセンサから3mとなっている．

3.7 問題点

3.5節のようにKinectを用いて特定の家電製品を操作・管理するシステムが導入され始めている．これにより家電の操作を直感的に行うことが可能となり老人や子どもにも使いやすくなっている．しかし家電の中には給湯器や電子錠などの操作を誤ると危険なものも存在する．このような家電の使用を制限するためには3.6節のような個人識別システムを組み込むことが望ましい．しかし前述のシステムは，距離画像センサのみを使用して特徴量を求めているため，同じ身長の人物を見分けることができない．これでは個人識別システムとして精度にかけるものとなってしまう．

この問題を改善するため，個人識別システムにセンサフュージョン技術を取り入れることを提案する．これは，複数のセンサからの情報を組み合わせることで計測の精度を向上させるという技術である．Kinectは初めから複数のセンサを搭載しているため，この技術を取り入れた際に問題が生じにくいのではないかと考えた．センサフュージョン技術を取り入れることでどの程度識別精度が向上するのかを，単一のセンサのみを使用した識別システムと性能評価の比較を行うことで検証する．

第 4 章　Kinect を用いた個人識別

4.1　概要

Kinectの利用法は多岐にわたるが, 家電リモコンのような家庭内での使用を想定している場合, 小さな子供の利用を制限したり利用者ごとにサービスを提供できるように個人識別を行えることが望ましい.

本研究では, Kinectに搭載されている複数のセンサを用いてセンサフュージョンを行うことで個人識別システムの識別精度向上を目指す. そのためにはまず, 複数のセンサから得られた情報をもとにそれぞれ別の特徴量を算出する. 次に, 特徴量を複数用いることでどの程度識別精度が向上するかの検証を行う. 今回は特徴量として, 深度画像からの骨格情報とRGB画像からの顔画像を用いた.

4.2　センサフュージョンとは

複数のセンサからの情報を組み合わせることで, 欠落した情報を相互に補完し計測の精度や信頼性を高める技術をセンサフュージョンという. Kinectには初めから複数のセンサが搭載されているため, この技術を用いたシステムの開発にとても適したデバイスであるといえる.

4.3　特徴量の算出

個人識別を行うためには, Kinectから得られた情報を用いて特徴量を算出しなければならない. センサフュージョンを行うために, 距離画像センサとカメラ画像センサから得た情報をもとに2種類の特徴量を算出する.

4.3.1　深度画像からの特徴量（骨格特徴量）

深度情報から得られる特徴量にはジョイント間の距離を利用する. 使用する部位はなるべく頭に近いものを選択する. これはカメラに上半身しか映っていなくても深度画像・RGB画像から特徴量を得ることができ, 探索範囲を限定することで処理を軽くすることが目的である. 事前実験として左右の上腕の長さ, 肩幅の長さ, 肩幅と胴体長さの比率の測定を行い, 最も数値のブレが小さくかつ個人差の出やすかった肩幅の長さを今回利用する特徴量とした. 第2章で述べたようにKinectは人物を認識すると, 各関節の座標を3次元のデータとして取得することができため, これを用いることで右肩と左肩の座標から肩幅を算出し骨格特徴量とした. 求め方を以下に示す.

$$\text{骨格特徴量}=\sqrt{(\text{右肩のX座標}-\text{左肩のX座標})^2+(\text{右肩のY座標}-\text{左肩のY座標})^2+(\text{右肩のZ座標}-\text{左肩のZ座標})^2}$$

4.3.2　RGB 画像からの特徴量（顔特徴量）

カメラ画像センサで得たRGB画像から，目・鼻・口などの顔部品がどの座標にあるかを検出しそれらの大きさや配置パターンを数値化したものを特徴量とする．今回座標データを得るために無料APIサイトである「detectFace();」[7]を利用する．このサイトは顔を含んだ画像を送ると顔部品の位置を認識し，顔を構成する特徴的な部分(以下特徴点)の座標データを送り返すAPIを無料で提供している．返された特徴点データには座標のほかに「信頼度」という項目がついており正しく顔を認識できているかの指標にすることできる．信頼度に対する閾値を設定することで識別精度を変化させることができるため今回のシステムに利用している．例えば，閾値を0.7に設定した場合，使用する特徴点の信頼度がすべて0.7以上でなければ特徴量算出の処理に進まず画像取得に戻るといった具合である．

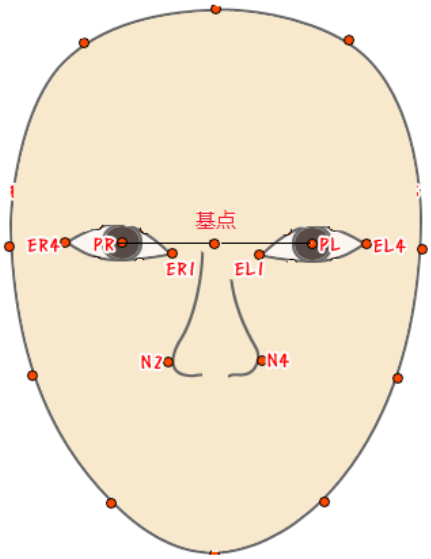

図8：使用した特徴点と名前

特徴点座標から特徴量を得る手法として，瞳間距離と基点-特徴点間距離の比率を使用している．瞳間距離とは右瞳(PR)と左瞳(PL)の距離であり，基点は瞳間距離の中点としている．この手法を用いることで得られる特徴量はユーザとカメラの距離および座標のずれによるノイズの影響が小さい．そのためより正確な数値を得ることができ識別精度の向上につながると考える．今回用いる特徴点は，目頭(ER1,EL1)・目尻(ER4,EL4)・鼻翼(N2,N4)とした．使用する特徴点が多くなると画像の読み込みに時間がかかるため，位置の変化が少ない箇所を選出している．特徴点の算出式は以下の通りである．

$$瞳間距離=\sqrt{(PRのX座標-PLのX座標)^2+(PRのY座標-PLのY座標)^2}$$

$$顔特徴量=\sum\frac{\sqrt{(特徴点のX座標-基点のX座標)^2+(特徴点のY座標-基点のY座標)^2}}{瞳間距離}$$

4.4　センサフュージョンを用いた個人識別システム

個人識別システムは4.3節の特徴量をそれぞれ登録し，登録した人物を認証するシステムである．本システムには人物を登録する「登録モード」，カメラ画像センサからの情報のみを用いて人物を識別する「RGB画像識別モード」，距離画像センサからの情報のみを用いて人物を識別する「深度画像識別モード」，センサフュージョンを用いて識別を行う「センサフュージョン識別モード」の4つのモードを搭載している．

登録モードでは，Kinectのセンサが人物を1人だけ認識した場合に特徴量を算出し，入力された名前とともに個人データとしてデータベースに保存する．

カメラ画像識別モード, 距離画像識別モードおよびセンサフュージョン識別モードでは，人物を認識し特徴量を算出した際に，登録モードで保存した特徴量データのうち対応しているものを読み取り比較を行う．特徴量が一致した場合は，一緒に保存していた名前を画面に表示し個人識別が成功したものとする. 一致するデータが存在しない場合は，登録されていないことをユーザに伝え新たに人間の認識待つこととなる．

4.5　開発環境

今回のシステムの開発環境を以下に示す.

表1：開発環境

OS	Windows7
開発ツール	Microsoft Visual Studio 2010
開発言語	C++
ライブラリ	OpenNI 1.5.2

OpenNIとはKinectのセンサ部を開発したPrimeSence社が中心となって開発したオープンソースのライブラリである．以下にOpenNIの仕様を示す.

表2：OpenNIの仕様

対応OS	WindowsXP, Vista, 7 Ubuntu Mac OS X
開発言語	C, C++, C#
RGBカメラ解像度	640×480
距離カメラ解像度	640×480
認識距離範囲	500-10000mm

第 5 章　実験と評価

5.1　概要

本章では, 単一の特徴量を用いたシステムと複数の特徴量を用いたシステムの認識精度を比較することで, センサフュージョン技術が個人認証システムにどの程度有効かを検証した.

5.2　実験手順

今回行った3つの実験の手順について示す.

手順①　データベース登録

Kinectに人物を認識させるために, Kinectのセンサ範囲内に上半身が入るように直立する．Kinectが人物を認識すると両肩･首･みぞおちの部分に赤いマーカーが描かれる．この状態の時に「s」キーを押ししばらく待機すると名前を入力するように求められる．名前を入力するとデータベースに名前と特徴量が登録される．今回の実験ではこれを5人分行った.

手順②　人物識別

識別モード移行キー(深度画像識別モード「d」，RGB画像識別モード「c」，センサフュージョン識別モード「f」)を押すことで対応するモードになる．これでKinectが人物を認識した際に, モードに応じた特徴量を使用してデータベースとのマッチングを行い登録した人物かを識別する．登録済みならば登録名を出力し，そうでないなら未登録であると出力する．登録している5人，登録していない5人の人物を正確に識別できるかを調査した.

手順③　性能評価

手順②の結果からそれぞれのシステムの性能を, 情報検索における性能評価指数の 1 つである適合率･再現率･F値で評価する．この3つの値は，最大値1，最小値0でありF値が1に近い程性能が良いことを意味している．求め方を以下に示す.

$$\text{適合率} = \frac{\text{登録済みの人物を認識した際の正答数}}{\text{登録済み人物を識別した際の正答数} + \text{未登録人物を識別した際の誤答数}}$$

$$\text{再現率} = \frac{\text{登録済みの人物を認識した際の正答数}}{\text{登録済み人物を識別した際の正答数} + \text{登録済み人物を識別した際の誤答数}}$$

$$\text{F値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}}$$

5.3 実験環境

実験は表3に示した環境で行った.

表3：実験環境

データベース登録人数	5人(A, B, C, D, E)
データベース未登録人数	5人(F, G, H, I, J)
測定回数	10回
測定距離	2m
Kinect設置高度	1.2m

5.4 骨格特徴量を用いた個人識別

5.4.1 実験目的

センサフュージョンを用いた個人認識システムの性能を評価するための比較対象として，骨格特徴量を使用した個人認識の精度を検証した．深度画像から得られる特徴量として，数値のブレが小さく個人差が比較的出やすい肩幅の長さを使用した．骨格特徴量を用いた場合は，体型に差がある人物を識別することが可能であると予想する．

5.4.2 実験結果

骨格特徴量のみを用いた場合に個人識別を行えるかの実験結果は表4のようになった. 表の括弧内の数値は登録した特徴量, 正答数はその人物が正しく識別された回数, 処理回数は特徴量を算出する処理を行った回数を表している.

表4：骨格特徴量による識別結果

名前(登録特徴量)	正答数	処理回数
A(95.801114)	7/10	1.0
B(99.065766)	8/10	1.0
C(90.726311)	0/10	1.0
D(90.328570)	0/10	1.0
E(110.853485)	10/10	1.0
F(96.035142)	0/10	1.0
G(117.358612)	10/10	1.0
H(100.149636)	0/10	1.0
I(92.275091)	4/10	1.0
J(95.120586)	0/10	1.0
	39/100	平均1.0

これらの表により骨格特徴量のみを用いて個人識別を行う場合，体格に差のある人物は識別することが可能であるが，CさんとDさんのように体格にあまり差が見られない人物を識別することが不可能であるという結果になった．また，Kinectは人物を認識している限り骨格情報を取得し続けることができるため，情報不足によって特徴量算出ができないといった事態が起こらず常に算出処理回数が1回であった．

また，以上の結果から求めた性能評価は表5のようになった．

表5：性能評価（骨格特徴量）

適合率	再現率	F値	平均処理回数
0.41	0.50	0.450	1.0

表5より適合率･再現率共に高い数値とは言えず，骨格特徴量のみを用いて個人を識別することは難しいという結果になった．

5.4.3 考察

以上の実験より骨格特徴量の長所と短所について考察する．まず，長所として処理回数の少なさがある．これによりKinectが人物を認識してから結果が表示されるまでの時間を短縮することが可能である．短所としては識別が不安定なことである．体格の似ている人物の識別は困難であり精度を向上させるためには他の要素を取り入れる必要がある．

5.5　顔特徴量を用いた個人識別

5.5.1　実験目的

センサフュージョンを用いた個人認識システムの性能を評価するための比較対象として，顔特徴量を使用した個人認識の精度を検証した． RGB画像から得られる特徴量を求めるための特徴点には意図せずに位置が変化することが少ない目頭･目じり･鼻翼を使用した．顔特徴量を用いた場合，特徴量を求める処理に時間がかかると予想する．信頼度閾値を変化させることで識別結果がどのように変化するかを計測した．

5.5.2　実験結果

RGB特徴値のみを用いて個人識別を行った場合の実験結果は表6～8のようになった．表の正答数とはその人物が正しく識別され画面に名前が出力された回数である．処理回数とは特徴量が算出されるまでに行われた，顔画像を取得する処理の回数を表している．信頼度閾値を0.9に設定した場合，該当する顔画像を取得することができず特徴量算出処理に入ることができないという結果になった．

表6：信頼度閾値0.6の顔特徴量による識別結果

名前	正答数	処理回数
A	6/10	7.2
B	5/10	9.1
C	3/10	5.5
D	3/10	6.3
E	7/10	7.3
F	6/10	10.1
G	5/10	6.0
H	5/10	5.6
I	7/10	6.9
J	2/10	7.6
	49/100	平均7.2回

表7：信頼度閾値0.7の顔特徴量による識別結果

名前	正答数	処理回数
A	7/10	12.3
B	7/10	10.8
C	5/10	7.4
D	5/10	10.3
E	8/10	9.0
F	5/10	11.9
G	6/10	12.6
H	5/10	15.1
I	8/10	10.6
J	4/10	10.0
	63/100	平均11.0

表8：信頼度閾値0.8の顔特徴量による識別結果

名前	正答数	処理回数
A	8/10	20.2
B	8/10	22.1
C	6/10	24.9
D	6/10	28.6
E	9/10	24.7
F	7/10	26.5
G	7/10	23.0
H	8/10	30.0
I	10/10	27.3
J	6/10	25.4
	75/100	平均25.3

これらの表より閾値を大きくするほど識別率の上昇につながっているが同時に処理回数も多くなっていることがわかる．特に閾値0.7と0.8を比較すると処理回数が14.3回も増加している．また，EさんとIさんの正答率が他の人より高いことから，この二人は特徴的な顔をしていることがわかる．

また，以上の結果から求めた性能評価は表9のようになった.

表9：性能評価（顔特徴量）

信頼度閾値	適合率	再現率	F値	平均処理回数
0.6	0.49	0.48	0.485	7.2
0.7	0.59	0.64	0.614	11.0
0.8	0.76	0.74	0.750	25.3
0.9	測定不能	測定不能	測定不能	測定不能

閾値を大きくすることで 3 つの値が確実に増加していることがわかる.

5.5.3 考察

以上の実験より顔特徴量の長所と短所について考察する．顔特徴量を用いた場合，識別精度の安定性が長所といえる. 信頼度の閾値を大きくすることで確実に正答率が上昇していることがわかる．しかし処理回数の上昇はそれ以上の欠点となっており，閾値0.7と0.8の処理回数に14.3回の差があることは見逃せない問題である．これではリアルタイムに識別することが難しい. 閾値を大きくすると処理回数が増加する理由として，KinectのRGBカメラの解像度があまり高くないため，閾値を満足する画像を得るのに時間がかかるためであると推測する.

5.6　センサフュージョンを用いた個人識別

5.6.1　実験目的

5.4節，5.5節で行った実験結果をもとにセンサフュージョンを用いた個人認識システムの性能の比較実験を行った．単一の特徴量での識別の欠点をうまく補い性能の向上が図れているのか検証した．同じ閾値での識別率･処理回数の比較によって性能が向上したかどうかを判断した．

5.6.2　実験手順

5.2節で作成，使用したデータベースを使用し個人を識別する．まず，処理の速い骨格特徴量を用いてデータベースの登録データを絞り込む．絞り込む方法はSQL文のWHERE句を用いて以下のようになっている．

SELECT * FROM face WHERE (骨格特徴量)*0.95 < value1 < (骨格特徴量)*1.05

そして，取得したRGB画像より得た顔特徴量と絞り込んだデータとの比較によって人物を識別する．今回の信頼度の閾値は，F値と処理回数のバランスが良かった0.7とし，表4，表7のデータとの比較を行った．これによりセンサフュージョンを行うことで識別率･処理回数がどのように変化したかについて調べた．また，実験結果から求めた性能評価について表5，表9のデータと比較を行う．

5.6.3　実験結果

センサフュージョンを用いた個人識別システムの実験結果は表10，11のようになった．表10は表7の正答数と今回の実験での正答数を比較したものである．表10は表7の処理回数と今回の実験での処理回数を比較したものである．正答数は9の増加，処理回数は0.7の増加という結果になった．

表10：正答数比較

名前	骨格特徴量の正答数	顔特徴量の正答数	センサフュージョンの正答数
A	7/10	7/10	8/10
B	8/10	7/10	7/10
C	0/10	5/10	5/10
D	0/10	5/10	5/10
E	10/10	8/10	10/10
F	0/10	5/10	5/10
G	10/10	6/10	10/10
H	0/10	5/10	7/10
I	4/10	8/10	9/10
J	0/10	4/10	6/10
	39/100	63/100	**72/100**

表11：処理回数比較

名前	骨格特徴量の処理回数	顔特徴量の処理回数	センサフュージョンの処理回数
A	1.0	10.8	10.6
B	1.0	12.3	12.4
C	1.0	12.6	11.1
D	1.0	10.3	13.0
E	1.0	9.0	10.6
F	1.0	11.9	11.4
G	1.0	7.4	9.9
H	1.0	15.1	14.1
I	1.0	10.6	13.5
J	1.0	10.0	10.5
	平均1.0	平均11.00	**平均11.7**

また，以上の結果から求めた性能評価を表 5，表 9 のデータと比較した結果は表 12 のようになった．並びは F 値の小さい順になっている．

表12：性能評価比較

	適合率	再現率	F値	平均処理回数
骨格特徴量	0.41	0.50	0.450	1.0
顔特徴量(閾値0.7)	0.59	0.64	0.614	11.0
センサフュージョン	0.73	0.70	0.715	11.7
顔特徴量(閾値0.8)	0.76	0.74	0.750	25.3

5.6.4　考察

同じ閾値での顔特徴量のみで識別を行った場合と比較し，確実に正答数が増加していることがわかる．特にEさん，Gさんの増加が著しい．これは，骨格特徴量による登録データの絞り込みによる成果であると考える．表4，5よりEさんとGさんは他の人に比べ骨格特徴量が特徴的である．そのため，骨格特徴量で絞り込みを行った結果，他の人物候補が消え正確な識別が行えたと推測できる．この点から識別率に関してセンサフュージョンを行うことは有効であったといえる．処理回数については，閾値を大きくしたわけではないため，変化が小さいのだと思われる．また，顔特徴量を算出した後に骨格特徴量の算出を行うため，2種類の特徴量を同時に用いた場合の処理化数は「顔特徴量の処理回数+骨格特徴量の処理回数」となったのだろう．これによりセンサフュージョンを行った場合に処理回数が急増するということがないことが明らかになった．

性能評価に関しては表12より，閾値0.8の顔特徴量識別には至らないものの近い値を得ることができた．処理回数を加味して評価すると上回る性能のシステムであるといえるであろう．

今回の実験では，骨格特徴量･顔特徴量が共に似通っている人物がいなかったため，精度の向上がはっきり見て取れた．今後の課題として，そのような人物にも対応するシステムを構築することが挙げられる．

第 6 章　まとめ

本論文では，センサフュージョンを用いた個人識別システムを提案した．提案手法のシステムの性能を評価するためにまず，単一の特徴量による個人識別の精度について検証した．その結果，骨格特徴量のみを用いた場合，処理回数に優れるが識別率の不安定さが目立った．顔特徴量のみを用いた場合，閾値を大きくするほど識別率も上昇するが，それ以上に処理回数が増してしまう．具体的には，識別率12%の上昇に対して処理回数は14.3回も増えてしまう．これではリアルタイムに識別することが難しい．手案手法では，骨格特徴量で登録データをあらかじめ絞り込み，顔特徴量の比較を行う回数を減らすことで，識別率9%の上昇に対し処理回数の上昇を0.7回に抑えることができた．これによりセンサフュージョンを用いて個人識別システムを構築することは有効であることが判明した．

今後の課題として，男女や大人子供の識別を行えるようにすることが考えられる．そのためには，もう一種類特徴量を取得できること望ましい．Kinectにはマイクセンサが搭載されているため，これを用いることで声の高低からより精度の高い識別システムの構築が可能であると推測する．

謝辞

本論文を作成するに当たり，ご指導頂いた三好力教授に感謝いたします．また，様々な助言を下さった三好研究室の皆様や，多忙であるにも関わらず実験データの集計に協力してくださった皆様に心から感謝したします．本当にありがとうございました．

参考文献

[1] “Kinect – Xbox.com”
<http://www.xbox.com/ja-JP/kinect>

[2] 「Kinectの技術を採用した障害者支援サービス”OAK” キッザニア東京で体験会実施」
<http://www.famitsu.com/news/201210/05022394.html>

[3] “wi-Goプロジェクト”
<http://vimeo.com/24542706>

[4] 「Kinectによる側弯症計測システム」
<http://k4wa.com/report/grandprix.html>

[5] 「三井ホーム、KinectでTV/ブラインド/照明が操作できる次世代住宅公開」
< http://pc.watch.impress.co.jp/docs/news/20120911_558720.html>

[6]

[7] “detectFace();”
< http://detectface.com/ >

[8] 中村薫(2011)『Kinectセンサープログラミング』秀和システム.

[9] 米谷和記，三好力，”センサフュージョンを用いた個人認証”，情報処理学会第75回全国大会(2013発表予定)

付録

```
#include<iostream>
#include<stdexcept>
#include<vector>
#include < vcclr.h >
#include<opencv/cv.h>
#include<opencv/highgui.h>

#include<XnCppWrapper.h>
#include"sqlite3.h"

const char* CONFIG_XML_PATH = "SamplesConfig.xml";
const char* FFACE_CASCADE_PATH =
"C:/OpenCV2.1/data/haarcascades/haarcascade_frontalface_alt2.xml";
const char* PFACE_CASCADE_PATH =
"C:/OpenCV2.1/data/haarcascades/haarcascade_profileface.xml";

int flag = 1;

using namespace std;
using namespace xn;
using namespace System;
using namespace System::Net;
using namespace System::Xml;
using namespace System::Drawing;
using namespace System::Windows;

using namespace System::Data::SQLite;

//ユーザ検出
void XN_CALLBACK_TYPE UserDetected(UserGenerator& generator,
XnUserID nId, void* pCookie){
cout << "ユーザ検出:" << nId << " " << generator.GetNumberOfUsers() <<
"人目" << endl;
generator.GetSkeletonCap().RequestCalibration(nId, TRUE);
}
//ユーザ消失
void XN_CALLBACK_TYPE UserLost(UserGenerator& generator,
XnUserID nId, void* pCookie){
cout << "ユーザ消失"<< endl;
flag=1;
}

//キャリブレーションの終了
void XN_CALLBACK_TYPE CalibrationEnd(SkeletonCapability&
capability, XnUserID nId, XnBool bSuccess, void* pCookie){
if(bSuccess){
cout << "キャリブレーション成功。ユーザ：" << nId << endl;
flag=0;
capability.StartTracking(nId);
}
else {
cout << "キャリブレーション失敗。ユーザ：" << nId << endl;
flag=1;
}
}

//RGBピクセルの初期化
inline XnRGB24Pixel xnRGB24Pixel( int r, int g, int b){
XnRGB24Pixel pixel = {r, g, b};
return pixel;
}

CvRect depsize(int dep, int w, int h){
CvRect rect;
rect.x = 0;
rect.y = 0;
rect.width = w*1500/dep;
rect.height = h*1500/dep;
return(rect);
}

char* StoC(String^ Sline){
pin_ptr<const wchar_t> wch = PtrToStringChars(Sline);
size_t convertedChars = 0;
size_t    sizeInBytes = ((Sline->Length + 1) * 2);
errno_t err = 0;
char *Cline = (char *)malloc(sizeInBytes);
err = wcstombs_s(&convertedChars, Cline, sizeInBytes, wch,
sizeInBytes);
if (err != 0)printf_s("wcstombs_s    failed!¥n");
return(Cline);
}


int main(int argc, char * argv[]){
IplImage* camera = 0;
int mode = 0, fusion = 0, save = 0, read = 0;
int id = 0;
double Ravr[10];
float Savr[10];

int status = SQLITE_OK;
char sqlCreate[] = "CREATE TABLE face ( "
"id INTEGER PRIMARY KEY,"
"name TEXT NOT NULL, "
"value1,"
"value2,"
"value3"") ";
char sqlInsert3[] = "INSERT INTO face (id, name, value1 ) "
"                    values (%d, '%s', %f)         ";

char sqlInsert2[] = "INSERT INTO face (id, name, value2 ) "
"                    values (%d, '%s', %lf)         ";
char sqlInsert[] = "INSERT INTO face (id, name, value1, value2, value3 )
"
"                    values (%d, '%s', %f, %lf, %lf)         ";

try{
//コンテキスト初期化
Context context;
XnStatus rc = context.InitFromXmlFile(CONFIG_XML_PATH);
if(rc!=XN_STATUS_OK)throw runtime_error(xnGetStatusString(rc));
//イメージジェネレータ作成
ImageGenerator image;
rc = context.FindExistingNode(XN_NODE_TYPE_IMAGE,image);
if(rc!=XN_STATUS_OK)throw runtime_error(xnGetStatusString(rc));
//デプスジェネレータ作成
DepthGenerator depth;
rc = context.FindExistingNode(XN_NODE_TYPE_DEPTH,depth);
if(rc!=XN_STATUS_OK)throw runtime_error(xnGetStatusString(rc));
//デプスの座標をイメージにあわせる
depth.GetAlternativeViewPointCap().SetViewPoint(image);
//ユーザ作成
UserGenerator user;
rc = context.FindExistingNode(XN_NODE_TYPE_USER,user);
if(rc!=XN_STATUS_OK)throw runtime_error(xnGetStatusString(rc));
//キャリブレーションにポーズが必要
SkeletonCapability skeleton = user.GetSkeletonCap();
//コールバック登録
XnCallbackHandle userCallbacks, calibrationCallbacks;
user.RegisterUserCallbacks(&::UserDetected, &::UserLost, 0,
userCallbacks);
skeleton.RegisterCalibrationCallbacks( 0, &::CalibrationEnd, 0,
calibrationCallbacks );
//トラッキング範囲指定
//skeleton.SetSkeletonProfile(XN_SKEL_PROFILE_ALL);//全身
skeleton.SetSkeletonProfile(XN_SKEL_PROFILE_UPPER); //上半身
//skeleton.SetSkeletonProfile(XN_SKEL_PROFILE_LOWER); //下半身
//skeleton.SetSkeletonProfile(XN_SKEL_PROFILE_HEAD_HANDS); //
頭と手
//カメラサイズのイメージ作成
XnMapOutputMode outputMode;
image.GetMapOutputMode(outputMode);
camera = cvCreateImage(cvSize(outputMode.nXRes,
outputMode.nYRes),IPL_DEPTH_8U,3);
//検出器ロード
CvHaarClassifierCascade* FfaceCascade =
(CvHaarClassifierCascade*)::cvLoad(FFACE_CASCADE_PATH,0,0,0);
//顔検出用ストレージ
CvMemStorage* Fstorage = ::cvCreateMemStorage();
bool isDetected = true;
//表示状態
bool isShowImage = true;
bool isShowUser = true;
for(int x = 0; x < 10; x++){
Ravr[x] = 0;
Savr[x] = 0;
}
```

```
while(1){
//ノード更新待ち
context.WaitAndUpdateAll();
//画像データ取得
ImageMetaData imageMD;
image.GetMetaData(imageMD);
//深度データ取得
DepthMetaData depthMD;
depth.GetMetaData(depthMD);
//ユーザデータ取得
SceneMetaData sceneMD;
user.GetUserPixels(0, sceneMD);
memcpy(camera->imageData, imageMD.Data(), camera->imageSize);
if(mode == 0 && flag == 0){
XnUserID users[15];
XnUInt16 userCount = 15;
user.GetUsers(users, userCount);
for (int i = 0; i < userCount; ++i) {
if ( !skeleton.IsTracking( users[i] ) )continue;
XnSkeletonJointPosition Posi[5];
XnPoint3D pt[5];
for(int j = 0; j < 5 ; j++){
if(j==0)skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_HEAD,Po
si[j]);
if(j==1)skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_NECK,Po
si[j]);
if(j==2)skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_TORSO,P
osi[j]);
if(j==3)skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_RIGHT_S
HOULDER,Posi[j]);
if(j==4)skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_LEFT_SH
OULDER,Posi[j]);
pt[j] = Posi[j].position;
depth.ConvertRealWorldToProjective( 1, &pt[j], &pt[j] );
if(j==0)continue;
else cvCircle( camera, cvPoint(pt[j].X, pt[j].Y), 10, cvScalar( 255, 0, 0 ),
-1 );
}
//頭-首
double HtoN = Math::Sqrt(Math::Pow((pt[0].X - pt[1].X), 2) +
Math::Pow((pt[0].Y - pt[1].Y), 2) + Math::Pow((pt[0].Z - pt[1].Z), 2));
//首-胴
double NtoT = Math::Sqrt(Math::Pow((pt[1].X - pt[2].X), 2) +
Math::Pow((pt[1].Y - pt[2].Y), 2) + Math::Pow((pt[1].Z - pt[2].Z), 2));
//肩幅
double StoS = Math::Sqrt(Math::Pow((pt[3].X - pt[4].X), 2) +
Math::Pow((pt[3].Y - pt[4].Y), 2) + Math::Pow((pt[3].Z - pt[4].Z), 2));
double rangeA = StoS;
double rangeB = StoS/NtoT;
printf("肩幅:%lf,肩幅/首-胴:%lf¥n", rangeA, rangeB);

XnSkeletonJointPosition Position;
skeleton.GetSkeletonJointPosition(users[i],XN_SKEL_HEAD,Position);
XnPoint3D pos = Position.position;
depth.ConvertRealWorldToProjective( 1, &pos, &pos );
int dep = pos.Z;
if(pos.X>50&&pos.X<590&&pos.Y>50&&pos.Y<430&&dep>200){
CvRect rect = depsize(dep,100,100);
rect.x = pos.X-rect.width/2;
rect.y = pos.Y-rect.height/2;
cvSetImageROI(camera, rect);
if(isDetected){
cvClearMemStorage(Fstorage);
CvSeq* Ffaces = ::cvHaarDetectObjects(camera, FfaceCascade, Fstorage,
1.2, 2, CV_HAAR_DO_CANNY_PRUNING, cvSize(20,20));
for(int i = 0 ; i < Ffaces->total ; ++i){
::cvCvtColor(camera, camera, CV_BGR2RGB);
//cvSaveImage("image.png", camera);

IplImage* image_resized = cvCreateImage(cvSize(rect.width * 2,
rect.height * 2), camera->depth, camera->nChannels);
cvResize(camera, image_resized, CV_INTER_CUBIC);
cvSaveImage("image.png", image_resized);
::cvCvtColor(camera, camera, CV_BGR2RGB);

WebClient^ wc = gcnew WebClient();
wc->Headers->Add("Content-Type", "image/png");
ServicePointManager::Expect100Continue = false;
printf("送信¥n");
array<Byte>^ RData =
wc->UploadFile("http://detectface.com/api/detect?m=2&f=0","image.png"
);
printf("送信完了¥n");
String^ RText = System::Text::Encoding::ASCII->GetString( RData );
XmlDocument^ xml = gcnew XmlDocument;
xml->LoadXml(RText);
try{
float sPR =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='PR']/
@s")->InnerXml);
float sPL =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='PL']/
@s")->InnerXml);
float sER4 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='ER4'
]/@s")->InnerXml);
float sEL4 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='EL4'
]/@s")->InnerXml);
float sER1 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='ER1'
]/@s")->InnerXml);
float sEL1 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='EL1'
]/@s")->InnerXml);
float sN2 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='N2']/
@s")->InnerXml);
float sN4 =
float::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='N4']/
@s")->InnerXml);

if(sPR<0.6||sPL<0.6){
printf("%f,%f¥n",sPR,sPL);
throw 10;
}
else{
System::Windows::Point ^PR = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='PR']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='PR']/@
y")->InnerXml));
System::Windows::Point ^PL = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='PL']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='PL']/@
y")->InnerXml));
System::Windows::Point ^ER4 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='ER4']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='ER4']/
@y")->InnerXml));
System::Windows::Point ^EL4 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='EL4']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='EL4']/
@y")->InnerXml));
System::Windows::Point ^ER1 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='ER1']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='ER1']/
@y")->InnerXml));
System::Windows::Point ^EL1 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='EL1']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='EL1']/
@y")->InnerXml));
System::Windows::Point ^N2 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='N2']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='N2']/@
y")->InnerXml));
System::Windows::Point ^N4 = gcnew
System::Windows::Point(int::Parse(xml->SelectSingleNode("/faces/face/f
eatures/point[@id='N4']/@x")->InnerXml),
int::Parse(xml->SelectSingleNode("/faces/face/features/point[@id='N4']/@
y")->InnerXml));

System::Windows::Point ^N1 = gcnew System::Windows::Point((PR->X +
PL->X)/2,(PR->Y - PL->Y)/2);

float pd = (float)Math::Sqrt(Math::Pow((PR->X - PL->X), 2) +
Math::Pow((PR->Y - PL->Y),2));//瞳間距離
```

```
float n1toel4 = (float)Math::Sqrt(Math::Pow((EL4->X - N1->X), 2) +
Math::Pow((EL4->Y - N1->Y), 2)) / pd;
float n1toer4 = (float)Math::Sqrt(Math::Pow((ER4->X - N1->X), 2) +
Math::Pow((ER4->Y - N1->Y), 2)) / pd;
float n1toel1 = (float)Math::Sqrt(Math::Pow((EL1->X - N1->X), 2) +
Math::Pow((EL1->Y - N1->Y), 2)) / pd;
float n1toer1 = (float)Math::Sqrt(Math::Pow((ER1->X - N1->X), 2) +
Math::Pow((ER1->Y - N1->Y), 2)) / pd;
float n1ton2 = (float)Math::Sqrt(Math::Pow((N2->X - N1->X), 2) +
Math::Pow((N2->Y - N1->Y),2)) / pd;
float n1ton4 = (float)Math::Sqrt(Math::Pow((N4->X - N1->X), 2) +
Math::Pow((N4->Y - N1->Y), 2)) / pd;

float sum = n1toel4 + n1toer4 + n1toer1 + n1toel1 + n1ton2 + n1ton4;
printf("%f¥n",sum);
Ravr[id] = rangeA;
Savr[id] = sum;
id++;
if(id==10){
id=0;

double Range = 0;
float Sum = 0;
for(int x = 0; x < 10; x++){
Range = Range + Ravr[x];
Sum = Sum + Savr[x];
}
Range = Range/10;
Sum = Sum/10;
printf("%f,%lf¥n",Sum,Range);
if(save==1){

sqlite3* db = 0;
char *dberror = 0;
int in = 1;
int rc = sqlite3_open("Face.db", &db);//DBを開く

status = sqlite3_exec(db, sqlCreate, 0, 0, &dberror);//テーブル生成
printf("名前を入力してください¥n");
String^ line1 = Console::ReadLine();
char* Cline1=StoC(line1);
char insert_record[1][256];
do{
sprintf( insert_record[0], sqlInsert, in, Cline1, Sum, Range, rangeB );
status = sqlite3_exec(db, insert_record[0], 0, 0, &dberror);    //データ挿入
in++;
}while(status != SQLITE_OK);
sqlite3_close(db);//DBを閉じる
printf_s("%sさんを登録しました。¥n", Cline1);
save = 0;
for(int x = 0; x < 10; x++){
Ravr[x] = 0;
Savr[x] = 0;
}
id = 0;
}
if(read == 1){
SQLiteConnection ^db = gcnew SQLiteConnection("Data
Source=Face.db");
db->Open();
float SumNum = 10;
double RangNum = 10;
String^ str = "SELECT * FROM Face
where"+Sum*0.95+"<value1<"+Sum*1.05;
Object^ name1;
Object^ name2;
SQLiteCommand ^Output = db->CreateCommand();
Output->CommandText = str;
SQLiteDataReader ^reader = Output->ExecuteReader();
while(reader->Read()){
if (Math::Abs(Range - reader->GetDouble(3)) <= RangNum) {
RangNum = Math::Abs(Range - reader->GetDouble(3));
name1 = reader[1];
}
}
if(RangNum > 0.1) printf("ななしさんです¥n");
else Console::WriteLine(String::Format("この人は{0}さんです
¥n",name1));
reader->Close();
db->Close();
read = 0;
flag = 1;
}
}
}
}
catch(...){
printf("error¥n");
}
}
}
cvResetImageROI (camera);
}
}
}
cvCvtColor(camera, camera, CV_BGR2RGB);
cvShowImage("KinectImage",camera);
//キーイベント
char key = cvWaitKey(10);

//終了する
if(key == 27){ //ESCキー
break;
}
if(key == 'm'){
if(mode==0){
cout << "Mode1"<< endl;
mode = 1;
}
else if(mode==1){
cout << "Mode2"<< endl;


mode = 2;
}
else if(mode==2){
cout << "Mode0"<< endl;


mode = 0;
}
}
if(key == 'f'){
if(fusion==0){
cout << "check mode"<< endl;
fusion = 1;
flag=0;
}
else {
cout << "save mode"<< endl;


fusion = 0;
}
}
if(key == 'z'){
flag = 0;
}
if(key == 's'){
save = 1;
}
if(key == 'r'){
read = 1;
}
if(key == 13){
cvSaveImage("image.png", camera);
printf("ENTER¥n");
}
}

//コールバック削除
user.UnregisterUserCallbacks(userCallbacks);
}
catch(exception& ex){
cout<<ex.what()<<endl;
}

cvReleaseImage(&camera);

return 0;
}
```