情報理工学特別研究報告書

題目

Azure Kinect を用いた 3 次元計測に関する研究

学生証番号　953481

氏名　國松 準矢

提出日　令和 5 年 1 月 27 日

指導教員　蚊野　浩

京都産業大学
情報理工学部

要約

3 次元計測技術の 1 つに，測定装置から対象物に光を照射し，対象物からの反射光の遅延時間に基づいて位置を計測する ToF（Time-of-Flight）あるいは LiDAR（Light Detection and Ranging）と呼ばれる手法がある． ToF や LiDAR は対象物表面の 1 点だけでなく，測定装置を中心とする多数の 3 次元座標を測定する．このようにして得る 3 次元座標の集合を 3 次元点群と呼ぶ．3 次元点群は低レベルの空間情報であり，このままでは高度な処理が難しい．例えば，対象物の形状を 3D プリンタで再現するには，複数の視点から撮影された 3 次元点群を正しく位置合わせした上で 1 つにまとめ，さらに，ポリゴンパッチなどで表現された 3D モデルに変換する必要がある．

本研究では Microsoft 社から提供されている，ToF 方式の 3 次元計測装置 Azure Kinect を用いる．まず，Kinect の前に物体と鏡，および照明光を遮る壁などを配置し，鏡に写る物体までの距離を Kinect を使って測定した．この実験の結果，鏡に写る物体像までの距離を測定する光の経路が複数存在し，経路によって測定される距離が異なることがわかった．

次に，黒い小物体を測定した．その結果，60cm の距離に置いた黒い小物体は幅 28mm 以上であれば点群化することが可能であった．また，物体の黒い色が背後の壁に写っていたり，測定した点群が流れたりする現象が発生した．小さい物体の色が背後の壁に写る原因は，撮影に用いる RGB カメラと Depth カメラの間に若干の距離があり，2 つのカメラ間で生じる視差によって見え方が変化したことが原因である．点群が流れる現象は物体の境界付近で発生する．これは，物体の境界付近では反射光が不安定になることが原因であると思われる．しかし，詳細な理由については解明することができなかった．

次いで，Azure Kinect で様々な方向から対象物を撮影して取得した複数の 3 次元点群を位置合わせし，3D モデルに変換するプログラムを開発した．対象物の概ねの位置合わせには成功していて，3D モデルの生成もできた．しかし，頭頂部や足の裏の点群情報がない箇所を埋めることができず，その部分だけが開いているモデルができてしまった．

目次

1章 序論

立体物の形状を測定したり，空間の立体的な広がりを計測したりする技術を 3 次元計測技術と呼ぶ．この技術にはさまざまな方法があり，それらの中で,測定装置から対象物に光を照射し，対象物からの反射光の遅延時間に基づいて位置を計測する手法を ToF（Time-of-Flight）あるいは LiDAR（Light Detection and Ranging）と呼ぶ．これらは，立体物の形状を測定する 3D スキャナや自動運転を実現するための距離画像センサーなどに実用化されている．なお，ToF と LiDAR の技術的な詳細については本論文の対象外であるが，用いる光源の種類などによっていくつかの方式がある．

本研究で用いる Azure Kinect[1]は ToF カメラと呼ばれる方式である．これは,距離を測定するシーン全体に対して光を照射し，その反射光を ToF センサと呼ばれる画像センサーで撮影する．

ToF カメラや LiDAR スキャナは何かしらの走査機能を持っているので，対象物表面の 1 点だけを測定するのではなく，同時に測定装置を中心とする多数の 3 次元座標を同時に測定する．このようにして得る 3 次元座標の集合を 3 次元点群と呼ぶ．1 回の撮影で得る 3 次元点群が，ToF カメラなどの測定装置から得る生データである．3 次元点群の基本的なデータは測定装置を中心とする 3 次元座標の XYZ 値であるが，測定装置によっては，反射光の強さやカラー画像の RGB 値を有する場合もある．

3次元点群は低レベルの空間情報であり，このままでは高度な処理が難しい．例えば,対象物の形状を 3D プリンタで再現するためには，複数の視点から撮影された 3 次元点群を正しく位置合わせした上で 1 つにまとめ，さらに，ポリゴンパッチなどで表現された３D モデルに変換する必要がある．蚊野研究室の過去の研究[2]では，Kinect v1 と PCL(Point Cloud Library)という 3 次元点群を処理するためのオープンソースソフトウェアライブラリを用いて立体物の 3 次元モデルを生成するという研究があった．しかし，その研究では，3次元点群の自動的な位置合わせに失敗しており，手作業による位置合わせが必要であった．

本研究ではまず，Azure Kinect の精度に関する基礎的な実験を行った．次いで，色々な立体物を撮影し，3次元点群を処理するための新しいオープンソースソフトウェアライブラリ Open3D を用いて，過去の研究ではできなかった全ての 3 次元点群の位置合わせを実現することを目標としている．以下 2 章では Azure Kinect や ToF カメラの概要，Open3D の概要と用いる機能の紹介，ICP(Iterative Closest Point)の概要と関連研究について述べる．3 章では Azure Kinect の精度に関する実験と実験結果について述べる．4 章では 3 次元モデルを生成する手法と生成したモデル，課題点を示し，5 章で結論を述べる．

2 章　3 次元点群の取得と処理の概要

2 章では，Azure Kinect や ToF カメラの概要，研究で用いる Open3D の機能に関することを説明する．

2.1　Azure Kinect と ToF カメラの概要

Azure Kinect は深度測定のできるカメラと様々なセンサーを一体化した開発者向けのデバイスで，Microsoft 社から提供されている．この装置には RGB カメラや 360° マイク，慣性測定装置(モーションセンサー)，ToF 方式の深度カメラが内蔵されている．深度カメラには狭い視野で遠くまで計測できる NFOV(Narrow Field of View)と，近い距離で広い視野を計測できる WFOV(Wide Field of View)の 2 つの動作モードがある．高度な AI センサーを搭載した先進的なコンピュータビジョンと音声モデルを構築するために Azure Kinect DK(Development Kit)という開発者キットがある．この開発環境は複数の SDK(Software Development Kit)で構成されている(図 1)．この SDK には，カメラやモーションセンサーなどのデバイスへのアクセスを制御したり，データストリームをファイルに記録したりする Azure Kinect Sensor SDK が含まれている．本研究では Azure Kinect Sensor SDK を利用する．

Azure Kinect は図 2 のように右の RGB カメラと左の Depth カメラでそれぞれの見える範囲を測っている．2 つのカメラは僅かに離れている．

ToF カメラは周囲に照明光を照射し，物体からの反射光が返ってくるまでの時間を測定し，光の速度を元に距離を計測するカメラである （図 3)．反射光の遅延時間を測定する画像センサになっているため，密な距離計測ができる．しかし，光源としてシーンの全体に照射する光を使用しているため，反射光は弱くなる．そのため，レーザー光線を用いる LiDAR のように遠方の距離を測定することには適さない．

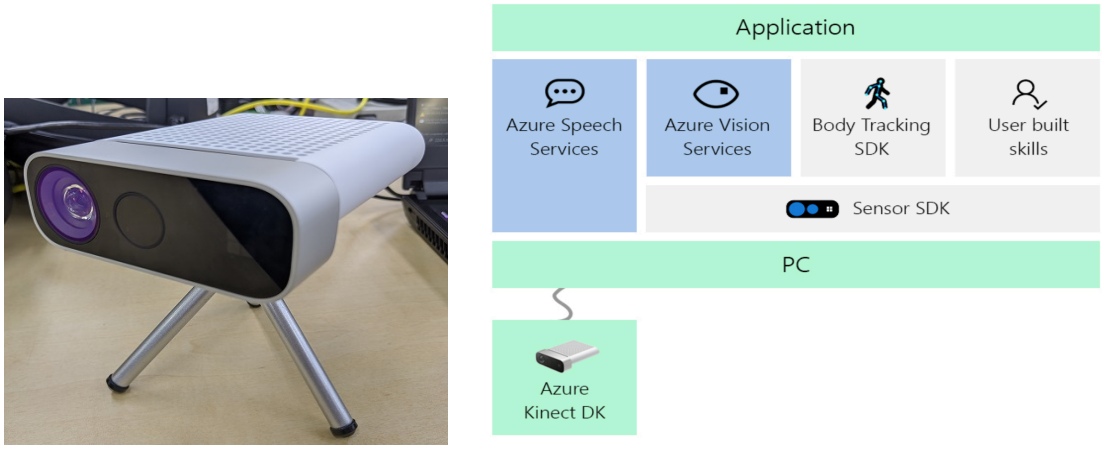


図 1　Azure Kinect の外観(左)と構成されている SDK(右)

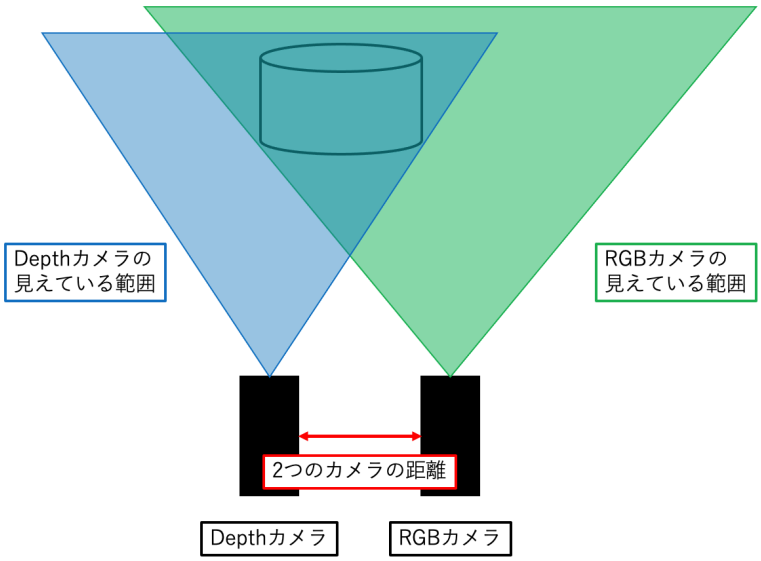


図 2　Azure Kinect の RGB カメラと Depth カメラの位置関係

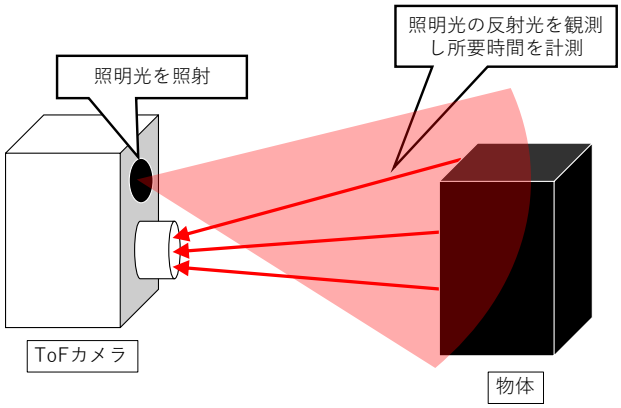


図 3　ToF カメラの模式図

2.2　Open3D の概要と今回の研究で用いる機能

Open3D は，3 次元データを扱うソフトウェアの迅速な開発を支援するオープンソースライブラリである[3]．様々な 3 次元データ処理関数を提供し，C++と Python から利用できる．OS は Linux，Mac，Windows で利用することができる．ただし，Azure Kinect に関する Open3D の機能は Mac では使用できない．本研究では，Azure Kinect に関するコードを Windows で使用し，それ以外の 3 次元データ処理は Mac で行う．

今回使用する主な機能は，外れ値除去とダウンサンプリング，法線推定，特徴点検出と特徴量の記述，RANSAC(RANdom SAmple Consensus)による姿勢推定，ポーズグラフの生成と最適化，ICP アルゴリズムである．この節では，外れ値除去とダウンサンプリ

ング，法線推定，特徴点検出と特徴量の抽出，ポーズグラフの生成と最適化，RANSAC による姿勢推定を説明し，ICP アルゴリズムは 2.3 で説明する．

深度カメラから取得した 3 次元点群データにはノイズが含まれている．特に，物体の境界部分では距離データが正しく計測できない場合が多く，物体境界から影のように後方に伸びる実際には存在しない点群が得られる．このようなノイズの中でも特に大きい誤差のものを外れ値と呼ぶ．これが点群の中に含まれていると，後述の ICP アルゴリズムなどの処理が失敗する可能性が高まる．それを防ぐために，前処理で外れ値除去を行う．Open3D には，統計的外れ値除去と半径外れ値除去の 2 つがある．本研究では半径外れ値除去を使用する．半径外れ値除去は各点を中心とする半径の球内の点を近傍点と見なした時，近傍点の個数が閾値未満となる点を除去するものである．

ダウンサンプリングは，3次元点群データを間引き，点の密度を均一にする処理である．3次元点群データの処理は多くの点を扱うため処理時間が膨大になることがある．そこで，点を間引くことで位置合わせなどを高速化することができる．

法線推定は 3 次元の幾何学的処理において重要な処理である．点群集合の一部の局所的な特徴量を抽出する時に，個々の点を表す特徴量として XYZ 値，RGB 値と共に法線の値が利用される．他には,点群の位置合わせを行う際に面と面を合わせる処理があるが，その際に法線の情報が必要になる．法線(法線ベクトル)は従来，直線や平面に対して定義されるものであり，点に対して定義できるものではない．点群の法線を求めるときは,暗にその点群が何らかの物体表面という曲面上に分布していることを仮定する．そして，各点の法線はその点における接平面に直交するベクトルとして定義する．点群の法線の求め方は,まず各点の近傍点を求める．そして，近傍点群の 3 次元座標に対して主成分分析を行う．主成分分析は近傍点群の分散共分散を求め，その固有値分解を行う手法である．通常の場合,次元削減を行うために主成分分析を行う場合は，固有値の大きいものから順に任意個の固有ベクトルを抽出する．固有値の大きい固有ベクトルはサンプルの分散が大きい軸を表す．本研究の場合は，法線ベクトルとして，近傍点群の分散が最も小さい軸を選ぶ．すなわち，最小固有値を持つ固有ベクトルを選べば，これが求める法線ベクトルとなる．Open3D にはこの手法で法線を求める関数がある．

特徴点は，点群の中でも特徴的なパターンを持ち，他の点との区別が付きやすいランドマークになりうる点を指す．これは点群位置合わせを行う際に重要になる．例えば，異なる視点から撮影した複数の 3 次元点群を位置合わせする時に，ある点群データから別の点群データへの座標変換を求めることになる．この時，点群データ間でどの点とどの点が対応するかを求める必要がある．ここで，一方の点群データ内の全て

の点に対する対応点をもう一方の点群データから探すことは計算量の観点から現実的ではない．このことから，点群位置合わせの最初の段階では，特徴点の間で対応点を探索する．特徴点を検出する手法には，物体表面の凹凸部分をもとに特徴点を検出する手法や等間隔にサンプリングした点を特徴点とする手法がある．本研究では等間隔にサンプリングした点を特徴点とする手法を用いる．

特徴量には，大きく分けて大域特徴量と局所特徴量がある．本研究では，局所特徴量の 1 つであり，Open3D に実装されている FPFH(Fast Point Feature Histograms)を使用する．局所特徴量は入力データの局所領域を表現する数値列である．多くの場合，あるデータと別のデータの特徴点の対応付けを行うため，特徴点の周辺領域を記述するために使用される．FPFH は，特徴点$\boldsymbol{p}$を中心とした小球領域に含まれる$\boldsymbol{k}$個の近傍点を求め，それらの点から 2 点を選ぶ全ての組み合わせ$(\boldsymbol{p}_i,\ \boldsymbol{p}_j)$に対して計算される下記のパラメータ$\alpha, \phi, \theta$のヒストグラムを計算する．これを高速化したものが FPFH である．$\boldsymbol{n}_i,\ \boldsymbol{n}_j$はそれぞれ点$\boldsymbol{p}_i,\ \boldsymbol{p}_j$の法線ベクトルである．

$$\alpha = \left((\boldsymbol{p}_i - \boldsymbol{p}_j) \times \boldsymbol{n}_i\right) \cdot \boldsymbol{n}_j$$

$$\phi = \frac{\boldsymbol{n}_i \cdot (\boldsymbol{p}_i - \boldsymbol{p}_j)}{\|\boldsymbol{p}_i - \boldsymbol{p}_j\|}$$

$$\theta = arc\tan\left(\left(\boldsymbol{n}_i \times \left((\boldsymbol{p}_i - \boldsymbol{p}_j) \times \boldsymbol{n}_i\right) \cdot \boldsymbol{n}_j\right), \boldsymbol{n}_i \cdot \boldsymbol{n}_j\right)$$

RANSAC による姿勢推定は，まず位置合わせを行う 2 つの点群(位置合わせ元点群をソース，位置合わせ先点群をターゲット)に対して，全ての対応点から予め決めておいた個数の対応点を選択し，幾何学的変換行列を算出する．次に，ソースを変換行列によって姿勢変換し，ターゲットとの距離を計算する．この値が予め決めておいた距離(マージン)より小さい場合はその対応点をインライア(外れ値ではないデータ)として判定する．また，インライアの対応点 1 点あたりの距離の平均値を計算する．この値が小さいほど，良い変換行列ということになる．この作業を繰り返して最も良い変換行列を最終結果とする．

ポーズグラフは，全ての 3 次元点群の位置と姿勢をグラフ構造化したものである．これにはノードとエッジという 2 つの要素がある．ノードは位置合わせする全ての 3 次元点群の集合$\boldsymbol{p}_i$の各位置姿勢行列を保持し，エッジは各位置合わせ元点群$\boldsymbol{p}_s$を各位置合わせ先点群$\boldsymbol{p}_t$に位置合わせした相対位置姿勢行列とそれから求めた情報行列を保持している．$\boldsymbol{p}_s$と$\boldsymbol{p}_t$の位置合わせを様々な組み合わせで繰り返してノードとエッジを追加していく． この時，ノードは位置合わせを行う点群同士が隣接している場合にノードを追加する．しかし，様々な組み合わせで位置合わせを行うため，エッジには誤

った位置合わせの組み合わせがある．これらがあると$\boldsymbol{p}_i$全てを正しく位置合わせして1つのオブジェクトにすることができない．そこで行うのがポーズグラフの最適化である．これで全てのエッジを考慮した元のポーズグラフを最適化し，誤った位置合わせをしたエッジを枝切りする．そして，残ったエッジをもとに各ノードの位置姿勢行列を修正する．Open3D には Levenberg-Marquardt 法という非線形最小二乗法を解く手法で最適化を行う関数がある．この最適化されたポーズグラフを用いて，位置合わせされた 3 次元モデルを生成する．

2.3 ICP の概要と関連研究

ICP は，3 次元データの活用において 3 次元点群同士の位置合わせ処理に広く用いられているアルゴリズムである．位置合わせ元点群となるソース点群$\boldsymbol{p}_s$と位置合わせ先点群となるターゲット点群$\boldsymbol{p}_t$を入力し，$\boldsymbol{p}_s$を$\boldsymbol{p}_t$に位置合わせするために必要な回転と並進(まとめて剛体変換$\boldsymbol{T}_p$)を推定する．ただし，ICP は初期値依存性のあるアルゴリズムである．$\boldsymbol{p}_s$と$\boldsymbol{p}_t$がある程度近い状態でないと正しい解は得られない．そのため,2.3 で述べた特徴点検出と特徴量の抽出,RANSAC による姿勢推定で粗い位置合わせを行い，これを ICP で扱う$\boldsymbol{p}_s$の初期姿勢とする．

ICP の手順を図 4 に示す．まず$\boldsymbol{p}_s$の各点と最も近い$\boldsymbol{p}_t$の点$\boldsymbol{x}$を対応させる．次に対応点同士の目的関数$\boldsymbol{E}$を定義し，$\boldsymbol{E}$が最も小さくなる剛体変換$\boldsymbol{T}_p$を推定する．目的関数には，対応点同士の距離の 2 乗和が最も小さくなるように$\boldsymbol{T}_p$を推定する Point-to-Point(下の式(1))と対応する点へのベクトルと$\boldsymbol{p}_t$の法線の内積が最も小さくなるように$\boldsymbol{T}_p$を推定する Point-to-Plane(下の式(2))がある．これらの式でκは対応点の集合，xは$\boldsymbol{p}_t$の点でpは$\boldsymbol{p}_s$の点である．$\boldsymbol{n}_x$は単位法線ベクトルである．

$$E(\boldsymbol{T}) = \sum_{(x,p)\in\kappa}\left(x - \boldsymbol{T}_p\right)^2 \qquad (1)$$

$$E(\boldsymbol{T}) = \sum_{(x,p)\in\kappa}\left(\left(x - \boldsymbol{T}_p\right)\cdot \boldsymbol{n}_x\right)^2 \qquad (2)$$

そして，$\boldsymbol{T}_p$によって$\boldsymbol{p}_s$を移動させた後の各点について，最も近い$\boldsymbol{p}_t$の対応点を選択する．これを元に 2 回目以降の探索を同様に行い，k回目の$\boldsymbol{E}$と$k+1$回目の$\boldsymbol{E}$の差が収束判定閾値γよりも小さくなった場合終了する．

関連研究として，金井廉ら[4]が部分運動のある剛体の位置合わせで ICP を元にした手法を提案していたり，山田皐平ら[5]が提案手法の位置合わせ評価の比較を行う時に，対象手法として ICP を用いたりしている．

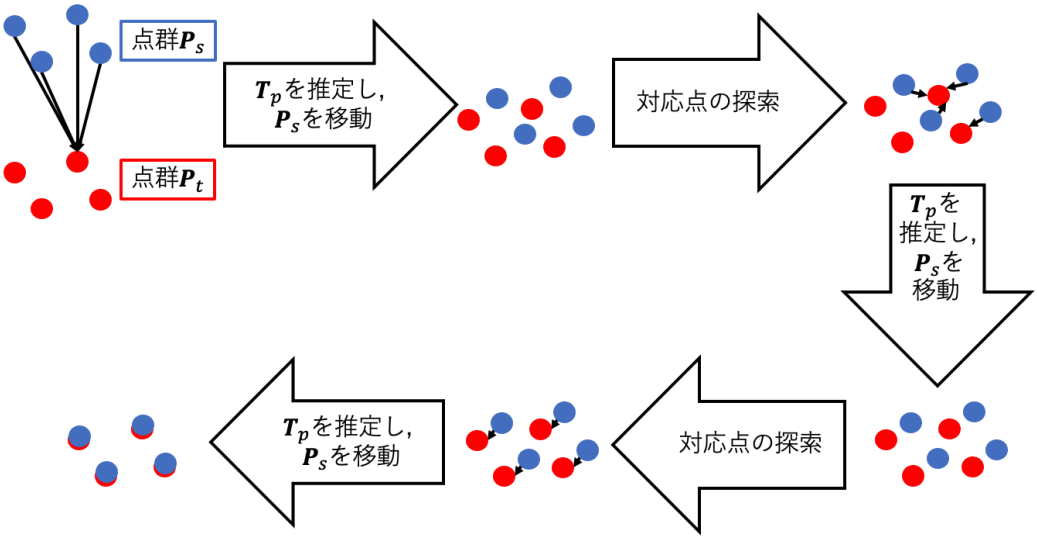


図 4　ICP アルゴリズムの動作

3 章　Azure Kinect の特性に関する基礎実験

3 章では，Azure Kinect を含めた ToF カメラの特徴とそれを踏まえた実験の内容を説明する．実験を行う Open3D を用いたプログラムは Python で動かしている．

3.1　ToF カメラの特徴

Azure Kinect を含めた ToF カメラは様々な形状のものを測ることができる．しかし，測定できないものや難しいものがある．例えば，光源として近赤外光(可視光に近い赤外光)を用いているため，透明な物体(ガラスや水など)は光が反射せず通過してしまうため計測できない．鏡や光沢の強い物体は光を鏡面反射して，センサーに光が戻ってこないため計測することができない(図5)．黒い物体に対しての測定も「黒色」が光を吸収してしまうため苦手である(図 6)．

図 5　鏡に映った物体の写真(左)とその 3 次元点群(右)

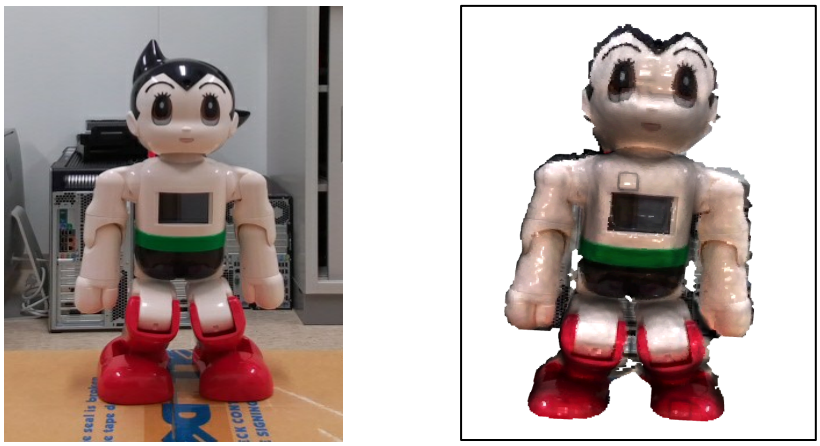


図 6　黒色を含む物体の写真(左)とその 3 次元点群(右)

しかし，図 5 を見ると鏡の部分は点群化されていないが，鏡に映った像が点群化されている．そして，図 6 を見ると髪にあたる部分は点群化されていないが，お腹部分やパンツの部分は点群化されている．

そこで本研究では，鏡に映った物体がどの位置に映っているのか，どのくらいの大

きさの黒色物体が点群化されるかを実験的に調べる．

3.2　鏡に映った物体像の測定

鏡に映る物体が Kinect Azure でどのように計測されるかを確認する実験を行った．図７および図８のように Azure Kinect 物体（サンタ），鏡を配置した．鏡に映る物体像までの距離を計測する光の経路は，Kinect -> 物体 -> 鏡 -> Kinect の経路と，Kinect -> 物体 -> Kinect の経路，Kinect -> 鏡 -> 物体 -> 鏡 -> Kinect の経路がある．このいずれの経路で距離が測定されているかについて実験した．

まず巻尺を使って，図７の①・②・③の距離を測った．物体から Kinect までの距離①が 71～72cm，Kinect から鏡までの距離②が 62～63cm，物体から鏡までが距離③が 60～61cm であった．次に，図 8 の状態で鏡の像までの距離を Kinect で測定した．また，物体と Kinect の間に壁を立てた状態（図 9）で，同じ像までの距離を測定した．

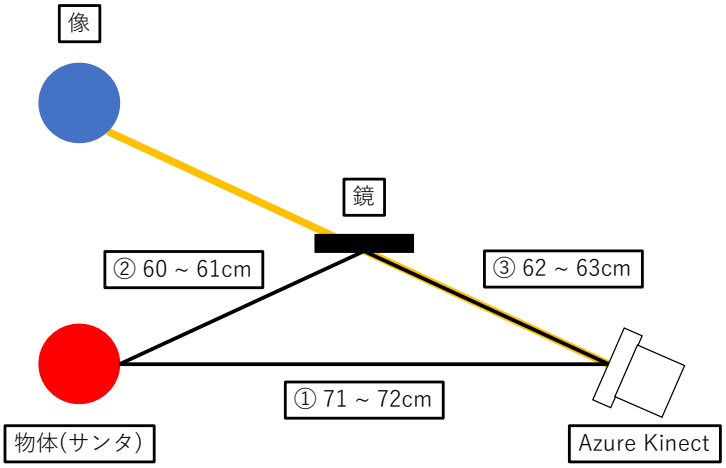


図７　物体や Kinect などの配置図

図８　撮影した時の様子

図 9　壁を置いて撮影した時の様子

図 8 の状態で測定した時，Kinect の RGB カメラで撮影した画像は図 10 左になった．また，3 次元点群を上から見下ろした感じでレンダリングしたものは図 10 右になった．図 9 の状態で測定した時，Kinect の RGB カメラで撮影した画像は図 11 左になった．また，3 次元点群を上から見下ろした感じでレンダリングしたものは図 11 右になった．図 10 右と図 11 右の図から，鏡の像に対する 3 次元点群の位置が異なることがわかる．すなわち，測定するための光が物体に直接当たる場合は手前側に測定され，測定するための光が鏡に反射して間接的に当たる場合は奥側に測定された．そして，図 10 の場合の像までの距離は 100cm，図 11 の場合の像までの距離は 123cm であった．100cm という距離は図 7 の（① + ② + ③）/ 2 に合致し，123cm という距離は（② + ③）に合致する．従って，図 8 の状態であれば，測定するための光は Kinect -> 物体 -> 鏡 -> Kinect の経路が支配的になり，図 9 の状態であれば，Kinect -> 鏡 -> 物体 -> 鏡 -> Kinect の経路を辿る光で距離を計測したことがわかった．

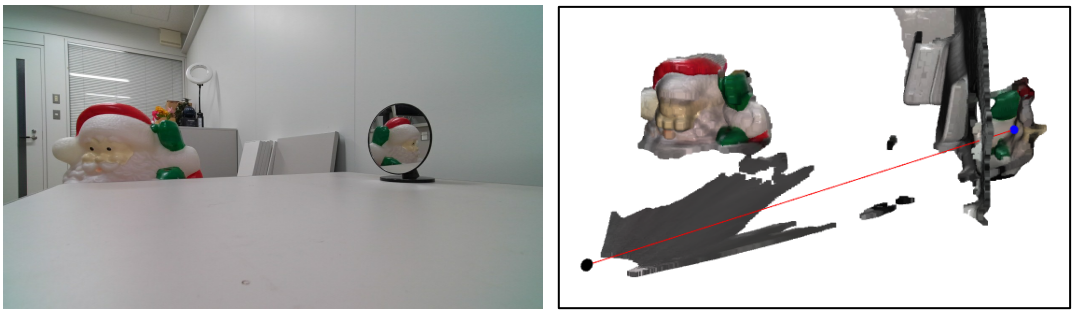


図 10　撮影した写真（左）と像から原点までの直線(右)

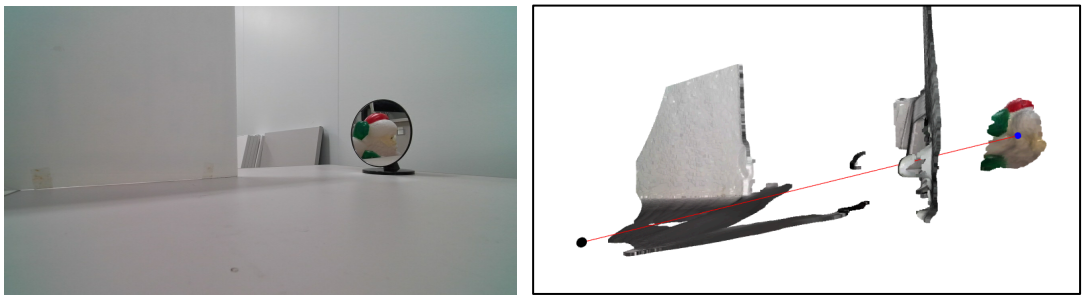


図 11　撮影した写真（左）と像から原点までの直線(右)

3.3　黒い小物体の測定

図 12 のように黒い小物体を Azure Kinect の前 60cm に置き測定する．この時，物体の背後 30cm の位置に壁を置いた．

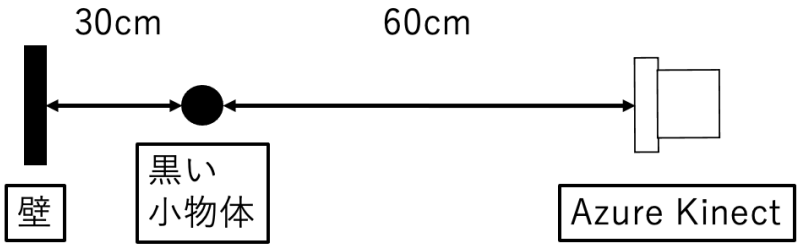


図 12　物体や Kinect などの配置図

黒い小物体は円柱型で，図 13 に示すように，左から割り箸(直径 5.5㎜)，ストロー(直径 10.9㎜)，サランラップの芯(直径 28.9㎜)，トイレットペーパーの芯(直径 41.8㎜)である．

図 13　撮影物体

図 14 のように撮影し，Open3D を使って３次元点群化し，描画して確認した．

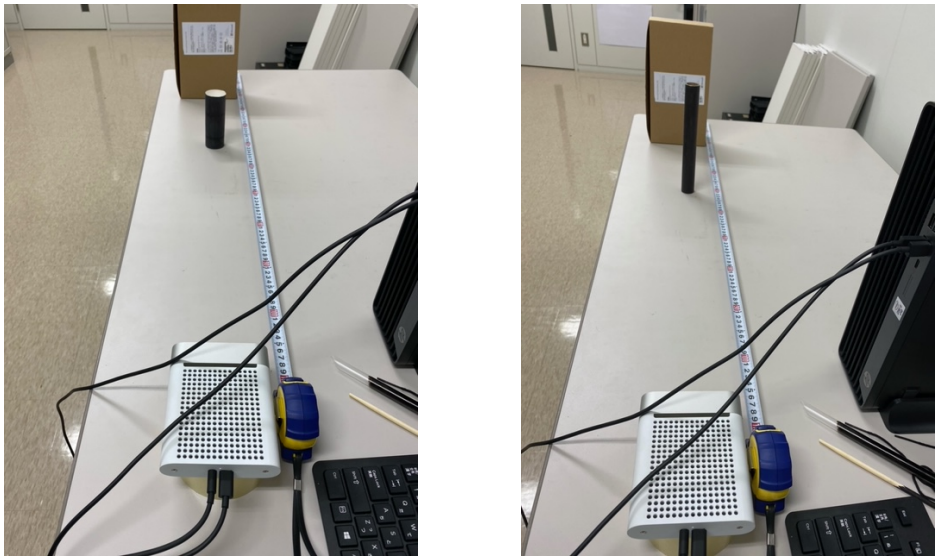


図 14　物体を撮影した様子(左:トイレットペーパーの芯，右:サランラップの芯)

これらの黒い小物体を撮影し，Open3D で可視化したものを図 15 から図 18 に示す．図の上側は撮影した様子を示し，左側はその様子の点群を正面からレンダリングした状態を示し，右側は点群を斜めからレンダリングし，物体の点群を見えやすくしたものである．図の緑枠は後ろの壁に映り込んだ黒色を示し，赤枠は黒い小物体の点群を示したものである．

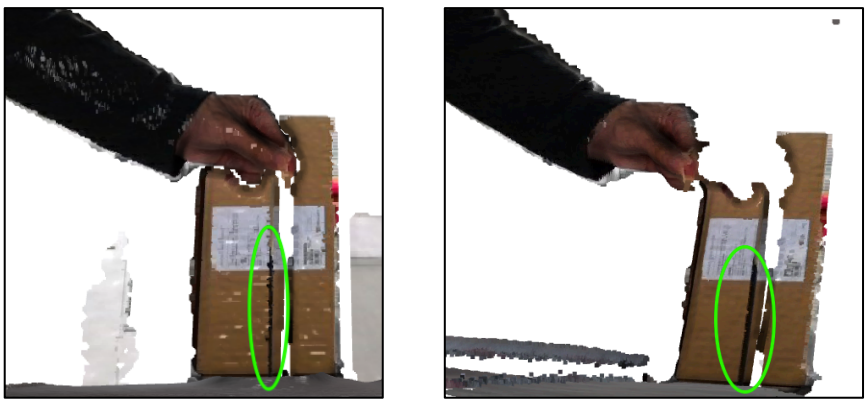
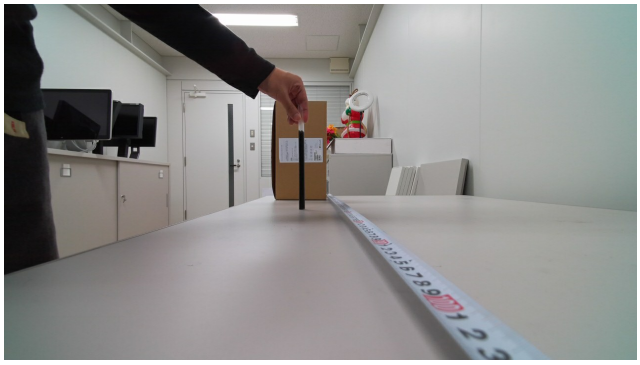
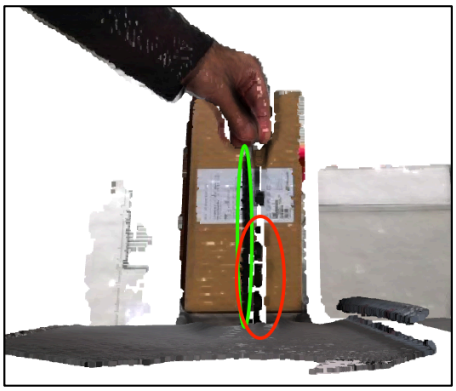


図 15　割り箸を点群化した結果





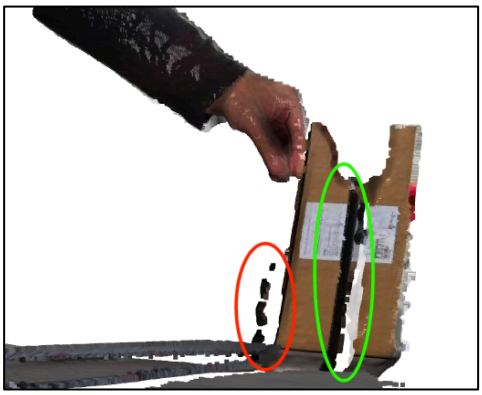
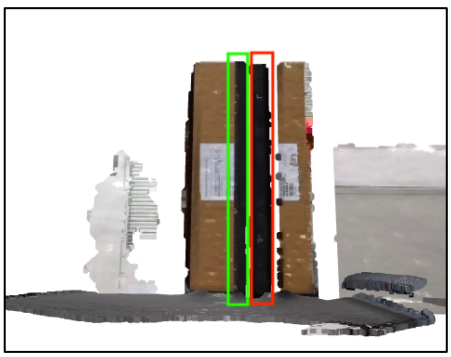


図 16　ストローを点群化した結果



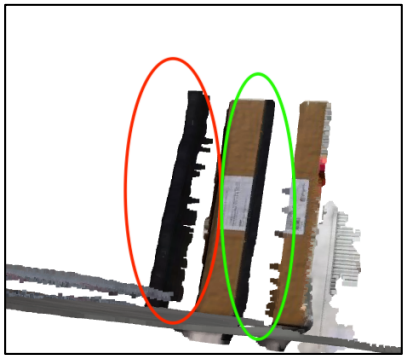


図 17　サランラップの芯を点群化した結果

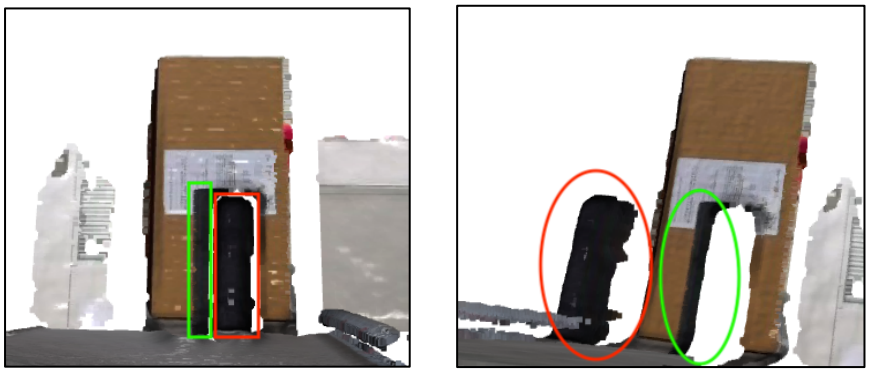


図 18　トイレットペーパーの芯を点群化した結果

結果を見ると，割り箸は点群化されず,色が後ろの壁に写っている．ストローは一部が点群化されている．点群化されていない部分は壁に色が写っている．サランラップの芯やトイレットペーパーの芯は表面部分が点群化されている．しかし，物体と壁の境界部分は斜めから見ると一部点群が流れたように点群化されていたり，壁に色が写っていたりしている．

このことから，60cm の距離においた黒い物体を測定する場合，28mm 以上の幅が必要であることがわかった．図 15～18 で，小物体の黒色が後ろの壁に映っている原因は，Kinect の表面で横に並ぶ RGB カメラと Depth カメラの間の若干の距離の存在である．図 2 のように対象物を測っているため，2 つのカメラで撮影したものには図 19 のように撮影距離やカメラ間距離によって視差(物体を見たときの見かけの位置ずれ)が発生する．すると，RGBD 画像(RGB 画像と Depth 画像のペアを統合した画像)を生成し，3 次元点群を生成した時に，視差によってそこには存在しない色情報が点群に現れてしまうことがある．このことから，黒色が後ろの壁に写っていると考えられる．

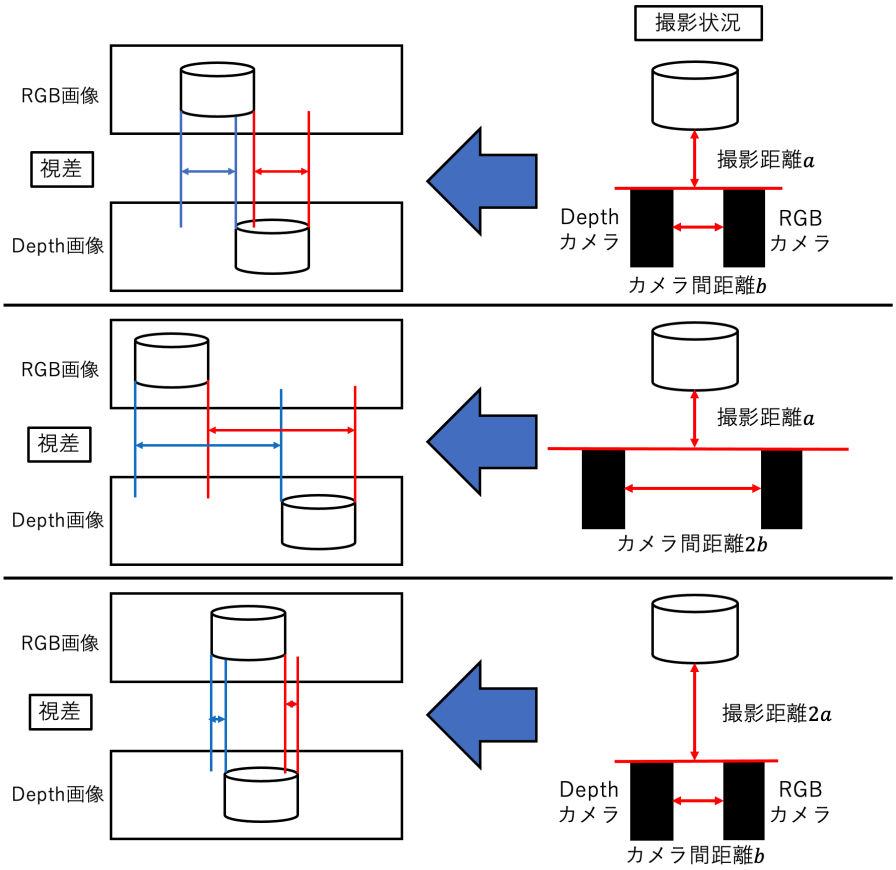


図 19　撮影距離やカメラ間距離によって変わる RGB 画像と Depth 画像の視差

図 17・18 右の上側の黒色点群で点群が流れるのは，物体の境界付近で反射光が不安定になることが原因である．物体の境界付近を照射した光の一部は物体上で反射する．一方，境界の外側を照射した光が物体の側面を照射する場合もある．また，物体の背後にある別の物体を照射する場合もある．このように複雑な現象が発生する可能性があり，その結果，測定される距離が不安定になると考えられる．

4 章　3 次元モデルの生成

4 章では，Azure Kinect で撮影した物体を Open3D で 3 次元モデルを生成する流れと生成したモデルについて説明する．モデル生成を行う Open3D を用いたプログラムは Python で動かしている．

4.1　実験に用いる物体

図 20 に示すプラスチック製のサンタクロース像を Kinect で撮影し，点群の位置合わせやメッシュの生成を行う．サンタクロース像は縦 76cm，横 35cm である．この物体の 3 次元モデルを生成することが目標である．

図 20　撮影したサンタクロース像

4.2　3 次元モデル生成の流れ

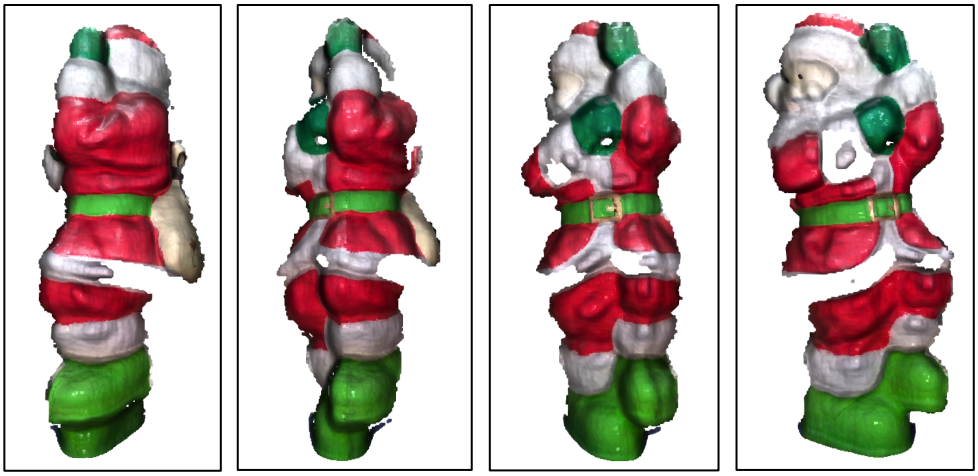
まず，Azure Kinect を用いて物体の全周を撮影して RGB 画像と Depth 画像，カメラパラメータを取得する．カメラパラメータには画像の解像度や焦点距離，光学中心などの情報が格納されている．1回の撮影では物体の一部しかわからないため，過去の研究[2]と同様に全周を 12 回分割して撮影する．この時，隣り合う位置での撮影に適度な重複領域を含ませる．この作業は Azure Kinect を用いるため，Windows で行う．それ以降の作業は Mac で行う．

取得した 12 個の RGB 画像と Depth 画像のペアから RGBD 画像(RGB 画像と Depth 画像のペアを統合した画像)を生成する．そして，RGBD 画像とカメラパラメータを用いて 3 次元点群を生成する(図 21 左)．生成した点群は，目的の物体以外の点群も含まれているため，物体以外の点群を取り除く．Open3D にマウスで指定した領域以外の点群を除去する関数があるので，それを使って物体の点群を切り取る(図 21 右)．できた 12 個の物体の点群(図 22)をファイルに保存する．この時のファイルは「ファイル名.ply」で保存した．

図 21　図 20 を点群化したもの(左)と物体のみを取り出した点群(右)

(0) 0°　(1) 30°　(2) 60°　(3) 90°

(4) 120°　(5) 150°　(6) 180°　(7) 210°



(8) 240°　(9) 270°　(10) 300°　(11) 330°

図 22　全周方向から撮影した物体の点群

生成した 12 個の物体の点群には，外れ値(図 23 左の赤丸内にある赤点)が含まれている．そのため点群に外れ値除去を行う．外れ値の多くは物体と背景の境界に生じている．外れ値除去を行ったのが図 23 右である．距離の段差が存在する境界では，その位置を正確に測定することが難しく，外れ値がボソボソと後方に伸びるように現れる．そして，外れ値除去した 12 個の点群をリストに格納する．

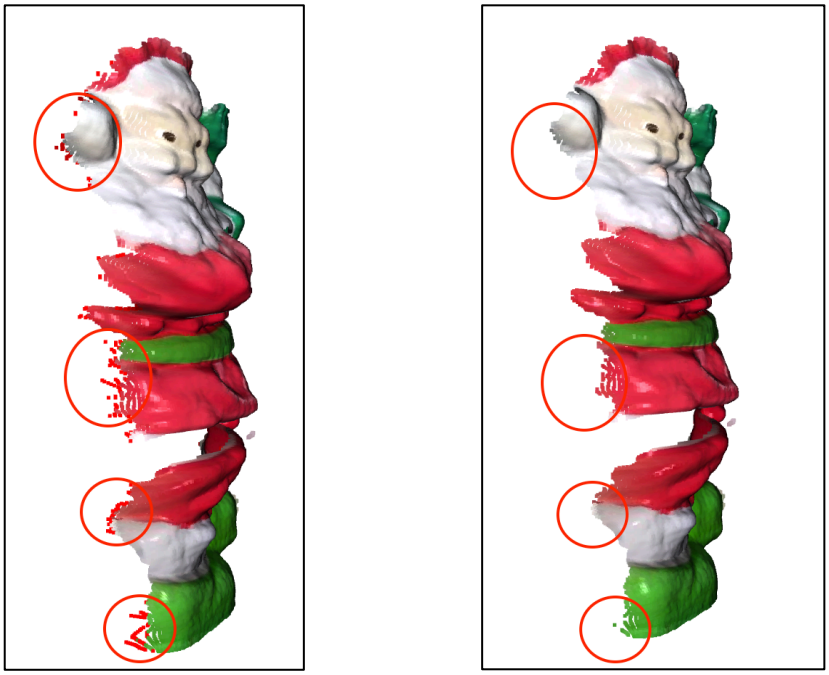


図 23　外れ値ありの点群（左)と外れ値除去した点群(右)

12 個の点群の位置姿勢を格納するポーズグラフを作成する．そして，12 個の点群の位置合わせを行い，ポーズグラフにノードとエッジを追加する．ただし，ノードは位置合わせを行なっている点群同士が隣接している時に追加を行う．この時の位置合わせは，ソース点群に対して，その後の点群全てをターゲット点群として 1 つずつ位置合わせを行う．例えば，ソース点群をリストの(3)番目とすると，ターゲット点群はその後のリストの(4)～(11)番目に格納されている点群に対して 1 つずつ位置合わせを行う．しかし，この位置合わせを行うと誤った位置合わせのエッジが出てくる．それはポーズグラフの最適化の際にそのエッジを取り除く．最適化は Levenberg-Marquardt 法という非線形最小二乗法を解く手法を使う．この最適化したノードを対応する点群に読み込み，3 次元モデルにする．

位置合わせは，最初に RANSAC による姿勢推定でソース点群の初期姿勢を行う．その後，初期姿勢のデータを元に ICP でターゲット点群との精密な位置合わせを行う．この時の位置合わせの目的関数は Point-to-Plane で行なった．

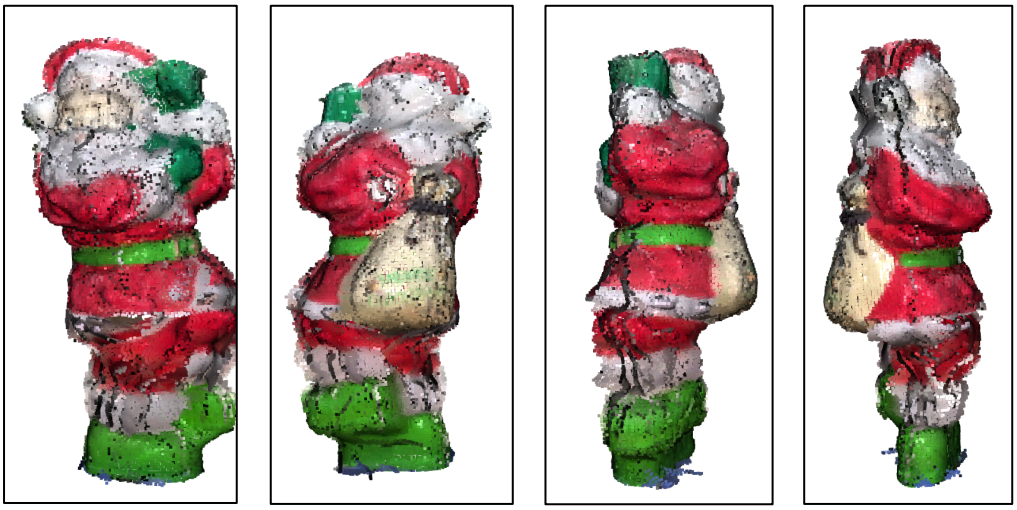
Open3D には RANSAC による姿勢推定を行う関数がある．これには 12 個の点群の特徴

点と特徴量が必要になる．特徴点の検出には 2.2 で述べた手法を用いる．すなわち，ダウンサンプリング処理を行なって残った点群を特徴点とする．この特徴点に対して FPFH 特徴量を算出した．

最後に位置合わせが終わった点群データにメッシュの貼り付けを行う．メッシュの貼り付け前に外れ値除去を行なって位置合わせ時に外れた点群を除去する．Open3D によるメッシュの貼り付けにはポアソン表面再構成という滑らかな曲面のメッシュを構成する関数がある．この関数を用いて表面構成を行う．ただし，Open3D でこの方法を使う場合は元の点群が法線を持っている必要がある．

4.3　3 次元点群モデルの確認

Open3D で位置合わせができた点群を図 24 に示す．全体的に位置合わせは概ね成功していて，図 20 のサンタクロースとわかるモデルとなっている．



(1)正面　(2)背面　(3)左側面　(4)右側面

図 24　位置合わせした点群

妥協点として，対象物を側面から撮影していたために，頭頂部や足の裏に当たる情報が少ない．そのため，位置合わせしたモデルは図 25 のように隙間が出てしまう．そして，位置合わせは点群全体を位置合わせする関係で図 26 のように一部位置合わせから外れてしまう点群が出てしまう．

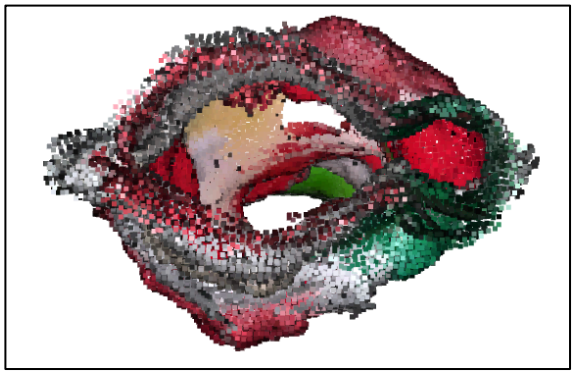


図 25　ICP の繰り返しによってできた隙間

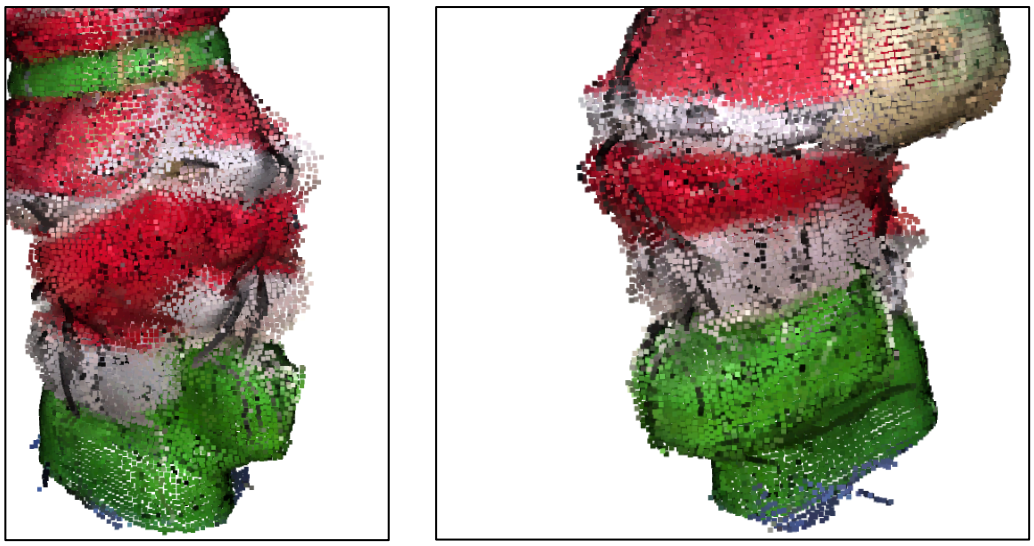


図 26　一部の位置合わせされていない点群

4.2　メッシュを貼り付けたモデルの確認と課題点

Open3D にある，ポアソン表面再構成という点群から滑らかな曲面のメッシュを生成する関数を用いて，位置合わせした点群にメッシュを貼り付けた結果を図 29 に示す．側面にメッシュが貼られている．

(1)正面　(2)背面　(3)左側面　(4)右側面

図 29　メッシュを貼り付けた結果

妥協点として挙げた，頭頂部(図 30)や足の裏の空間へのメッシュの貼り付けができなかった．これを解決するために，できたメッシュを新たに点群にサンプリングして再びメッシュの貼り付けを行なったが，それでもこの隙間は埋まらなかった．

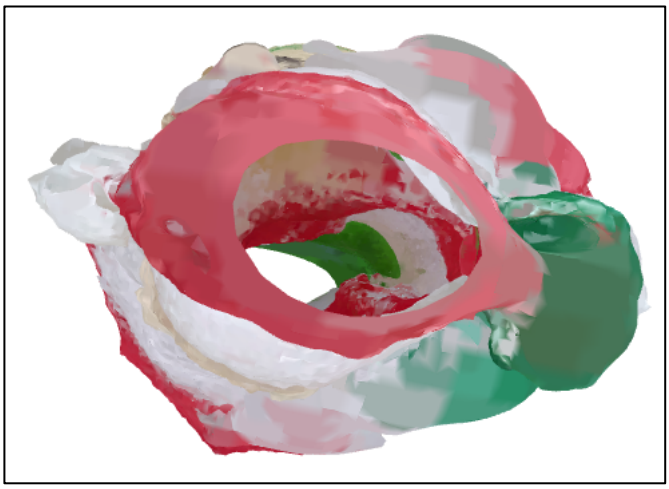


図 30　頭頂部のメッシュ

5章　結論

本研究では，Azure Kinect を用いて鏡に映った物体像と黒い小物体を測定することで，Azure Kinect の基本的な特性を明らかにした．また，立体物の全周を分割して測定した複数の点群を合成することで，全体の 3 次元モデルを生成するプログラムを作成した．

鏡に映った物体像の 3 次元的な位置を測定する光の経路は，その物体が Kinect から直接見えている時は Kinect -> 物体　-> 鏡 -> Kinect の経路で測定していた．Kinect から直接見えない時は Kinect -> 鏡 -> 物体 -> 鏡 -> Kinect の経路で測定していた．

黒い小物体を 60cm の位置で撮影する場合，直径 5.5mm のものは点群化されず，直径 10.9mm の場合に一部が点群化された．そして，28mm 以上のものであれば，安定して点群化された．また，物体の色がその背後の面に写るという問題と，物体の境界付近で点群の一部が流れたようになる現象が観察された．物体の色が背後の面に写る原因は，Kinect の RGB カメラと Depth カメラの間の若干の距離の存在である．この若干の距離と撮影距離の関係によって視差が発生し,RGBD画像にした時に視差によってずれた位置に色が付いたと考えられる．点群が物体の境界付近で流れる現象は，そこでの反射光が不安定になることに原因があると推察される．しかし，詳細な理由を明らかにすることはできなかった．

3 次元モデルの生成は目標であったサンタクロースの位置合わせに成功したが，頭頂部や足の裏の点群情報がない箇所をメッシュの貼り付けで埋めることができなかった．そのため，位置合わせの手法やメッシュの貼り付けも検討が必要である．

参考文献

[1] Azure Kinect DK のドキュメント,
https://learn.microsoft.com/ja-jp/azure/kinect-dk/

[2] Kinect と PCL を用いた立体物の 3 次元モデリング,
http://www.cc.kyoto-su.ac.jp/~kano/pdf/study/student/2013YamamotoPaper.pdf

[3] Open3D A Modern Library for 3D Data Processing, http://www.open3d.org

[4] 金井 廉, 藤田 悟, 「ICP を用いた位置合わせによる 3 次元点群の動き推定」, 情報処理学会第 77 回全国大会, pp.523-524, 2015 年.

[5] 山田 皐平, 山下 隆義, 藤吉 弘亘,「セマンティックセグメンテーションを用いた三次元点群の位置合わせ」,
http://mprg.jp/data/MPRG/F_group/F20181207_yamada.pdf

謝辞

本論文を作成にあたり,丁寧な御指導を賜りました蚊野浩教授に感謝いたします.

付録　本研究で開発したプログラムや関数

1. Azure Kinect 関連で使用したサンプルプログラム

azure_kinect_recorder.py

RGB 画像と Depth 画像ストリームを MKV ファイルに保存するプログラム.

azure_kinect_mkv_reader.py

MKV ファイルの RGB 画像と Depth 画像をフレームに変換し, カメラパラメータを取得するプログラム.

2. 外れ値除去や位置合わせで参考に使用したファイル

interactive_visualization.py

ジオメトリの切り取りや手動位置合わせなどを行うプログラム.

point_cloud_outlier_removal_radius.py

点群のダウンサンプリングと半径外れ値除去を行うプログラム.

multiway_registration.py

複数の点群の精密位置合わせを行うプログラム.

3. Open3D で使用した関数

read_point_cloud(filename)

ファイルから点群を読み込む関数.

filename : 読み込むファイルのパス.

read_image(filename)

ファイルから画像を読み込む関数.

filename : 読み込むファイルのパス.

write_point_cloud(filename, pcd)

点群をファイルに書き込む関数.

filename : 書き込むファイル名. 本研究ではファイル名.ply として保存した.

pcd : 書き込む点群オブジェクト.

write_triangle_mesh(filename, pcd)
メッシュをファイルに書き込む関数.
filename : 書き込むファイル名.
pcd : 書き込むメッシュオブジェクト.
draw_geometries(geometry_list)
点群やメッシュを可視化する関数.
geometry_list : 可視化するジオメトリのリスト.

read_pinhole_camera_intrinsic(filename)
ファイルからカメラ行列を読み込む関数.
filename : 読み込むファイルのパス.

create_from_color_and_depth(color, depth, convert_rgb_to_intensity=bool)
RGB カラー画像と Depth 画像のペアから RGBD 画像を生成する関数.
color : RGB カラー画像.
depth : Depth 画像.
convert_rgb_to_intensity : RGB 画像を輝度画像に変換するか. 本研究では false にしている.

create_from_rgbd_image(image, intrinsic)
RGBD 画像とカメラパラメータによって, 3 次元点群を生成する関数.
image : RGBD 画像.
intrinsic : カメラパラメータ.

remove_radius_outlier(nb_points, radius)
与えられた半径の球の中で nb_points 個未満となる点を削除する関数.
nb_points : 近傍点の個数の閾値.
radius : 球の半径.

voxel_down_sample(voxel_size)
voxel_size を 1 辺の長さとしたボクセルを等間隔にサンプリングした点群を出力する関数.
voxel_size : ボクセルサイズ.

estimate_normals(
search_param = open3d.geometry.KDTreeSearchParamHybrid(radius, max_nn))
法線を計算する関数.
search_param : 近傍探索のための KDTree 探索パラメータ.
radius : 球の半径.
max_nn : 近傍点の最大個数.

orient_normals_towards_camera_location(camera_location)
点群の法線を方向付ける関数.
Camera_location : 法線を向ける 3 次元座標

compute_fpfh_feature(input,
search_param = open3d.geometry.KDTreeSearchParamHybrid(radius, max_nn))
FPFH 特徴量を計算する関数.
input : 特徴量を計算する点群.
search_param : 近傍探索のための KDTree 探索パラメータ.

registration_ransac_based_on_feature_matching(source, target,
source_feature, target_feature, mutual_filter, max_correspondence_distance,
estimation_method, ransac_n,
checkers, criteria =
open3d.pipelines.registration.RANSACConvergenceCriteria(max_iteration,
confidence))
対応点探索と RANSAC による姿勢計算を行う関数.
source : 位置合わせ元点群.
target : 位置合わせ先点群.
source_feature : 位置合わせ元点群の特徴量.
target_feature : 位置合わせ先点群の特徴量.
mutual_filter : ソースポイントの対応関係が自分自身となるような相互フィルタリングを有効にするもの. 本研究では True としている.
max_correspondence_distance : インライアと判定する距離(マージン)の閾値.
estimation_method : 位置合わせ方法. 本研究では Point-to-Point による推定を行う TransformationEstmationPointToPoint としている.

ransac_n ： 姿勢変換行列の計算のためにサンプリングする対応点の個数．本研究では 3 としている．
checkers：2 つの点群の位置合わせが可能かを調べる．本研究では 2 つの判定関数で調べて当てはまらないものは除外する．
1 つ目は CorrespondenceCheckerBasedOnEdgeLength というソース点群内とターゲット点群内に個別に描かれた任意の 2 つの頂点が形成する線の長さが対応している点群内で類似しているかを確認する．
2 つ目は CorrespondenceCheckerBasedOnDistance という変換行列によってサンプリングした対応点 3 点を変換し，その距離が閾値よりも小さいかを確認する．
criteria ： RANSAC の終了条件．反復回数が max_iteration に達するか，検証中に測定されたフィットネスが，ある程度の確信を持ってアルゴリズムを早期に終了させることができることを示唆した場合に停止する．
max_iteration ： 反復が停止するまでの最大反復回数．
confidence ： 成功確率の希望値．早期終了を推定するために使用する．

registration_icp(source, target, max_correspondence_distance, init, estimation_method)
source ： 位置合わせ元点群．
target ： 位置合わせ先点群．
max_correspondence_distance ： 2 つの点群を対応付けする時の最大距離．
init ： source の初期姿勢．
estimation_method ： 位置合わせ方法．本研究では Point-to-Plane による推定を行う TransformationEstmationPointToPlane としている．

create_from_point_cloud_poisson(pcd, depth)
ポアソン表面再構成に基づいてメッシュを生成する関数．
pcd ： メッシュを貼り付ける点群．
depth ： 表面再構成に使用するツリーの最大深度．値が大きいほどより詳細なメッシュを貼り付ける．