

コンピュータ工学特別研究報告書

題目

ARKit の VIO 機能を利用したアプリ
JustBox の開発

学生証番号 744393

氏名 北野晴之

提出日 令和3年2月8日

指導教員 蚊野 浩

京都産業大学
コンピュータ工学部

要約

ARは、人間と実世界のインタラクションをコンピュータを使って拡張・強化する技術の総称である。その中でも、カメラで撮影した実写映像を3次元空間として認識し、そこに違和感なく3DCGを合成することで、あたかもそこにモノがある様に見せる技術が注目を集めている。このようなARを開発するためのプラットフォームとして、Apple社が提供するiOSフレームワークの一つであるARKitがある。

本論文ではARKitのVIO (Visual Inertial Odometry, カメラと慣性センサを使った空間測定) 機能を利用したアプリ Just Box を開発した。JustBoxは「物体をJust (ぴったり) に囲むBox (箱)」という意味で、測定物を収めるぴったりの直方体を画面に表示し、その箱の大きさをユーザに伝えることができるアプリである。プログラムは、ARKitを使ってPoint Cloudを取得する。Point Cloudは物体の形状を表現する3次元点群である。次いで、取得したPoint Cloudを、判別分析法を用いて測定物体の表面にあるPoint CloudとそうではないPoint Cloudに2クラス化する。最後に、物体表面にあると推定したPoint Cloudを収める最小の直方体を求めて、画面に表示する。

結果として、JustBoxで4種類の測定物を測定し、これらの物体をJustに収める直方体を描画することができた。しかし、Point Cloudをうまく取得することができず、Justな直方体を計算することに失敗することもあった。その理由の一つは、カメラを動かした際に、消えたり増えたりするPoint Cloudが残像として反応することであった。また、別な問題として、最初に取得したPoint Cloudから測定物体表面のPoint Cloudを抽出する処理が正しく動作しない場合もあった。さらに、現在の直方体の大きさの計算方法が不十分であることがわかった。

目次

1 章 序論	. . . 1
2 章 ARKit の特徴とプログラムの構成	. . . 3
2.1 AR と ARKit の位置付け	. . . 3
2.2 VIO による自己位置推定と Point Cloud の取得	. . . 4
2.3 ARKit を使ったプログラムの構成	. . . 5
2.4 3D オブジェクトの描画	. . . 7
3 章 JustBox の開発	. . . 10
3.1 JustBox について	. . . 10
3.2 JustBox の処理の流れ	. . . 10
3.2.1 測定物以外の Point Cloud の除去	. . . 11
3.2.2 Point Cloud を囲む直方体の求め方	. . . 13
4 章 実験結果	. . . 15
4.1 アプリの動作検証	. . . 15
4.2 サンプル数を増やした検証	. . . 16
4.3 課題	. . . 18
4.3.1 Point Cloud の2クラス化に失敗する場合	. . . 18
4.3.2 体積最小の直方体が軸平行でない場合	. . . 18
5 章 結論	. . . 20
参考文献	. . . 21
謝辞	. . . 21
付録	. . . 22

1 章 序論

AR (Augmented Reality, 拡張現実感) を用いることで, 実世界にさまざまなデジタル情報を付加して提供することができる. AR 技術は大きく成長している分野であり, ゲームやアプリケーションに取り込まれ, 多方面で活躍をしている.

AR アプリの例として「Pokemon GO」(図 1.1) が挙げられる. これは, 現実世界の中でポケモンが本当にトレーナーの目の前にいるように見えるアプリである. 最新のバージョンでは目の前に見えるだけでなく, ポケモンを見下ろしたりポケモンの真横に行ったりすることができる. Pokemon GO 以外にも, 位置情報とキャラクターなどを組み合わせるアプリが数多く登場し, AR ベースのゲームやアプリが人気コンテンツになっている.

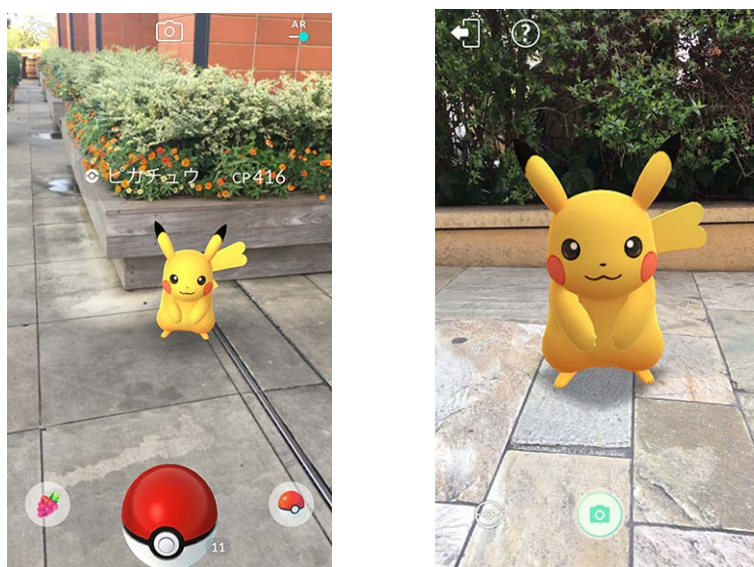


図 1.1 Pokemon GO の AR 使用例 (Pokemon GO 公式サイトより)

実用的な AR アプリとして, カメラを向けるだけで, 撮影された物体の長さや面積を測ることができる計測アプリがある. 代表的なものは Apple 社が開発したもので, iOS12 以上の iPhone や iPad に「計測アプリ (ARMeasure)」としてプリインストールされている.

図 1.2 に計測アプリで実際に物体を測った時の様子を示す. 測りたい物体にカメラをかざし, アプリの指示に従って操作するだけで, 辺の長さや面積を自動的に計算する. 机の上に置いてある物や, 壁にかかっている額縁のサイズであれば, 概ね正確なサイズを計測することができる. カメラで撮影した 1 枚の画像だけから正確な寸法を求めることは, 原理的には不可能なはずである. これが可能になっている理由は, iOS に

VIO (Visual Inertial Odometry, カメラと慣性センサを使った空間測定) 技術が組み込まれているからである。VIO は AR を実現する重要な要素技術の一つである。

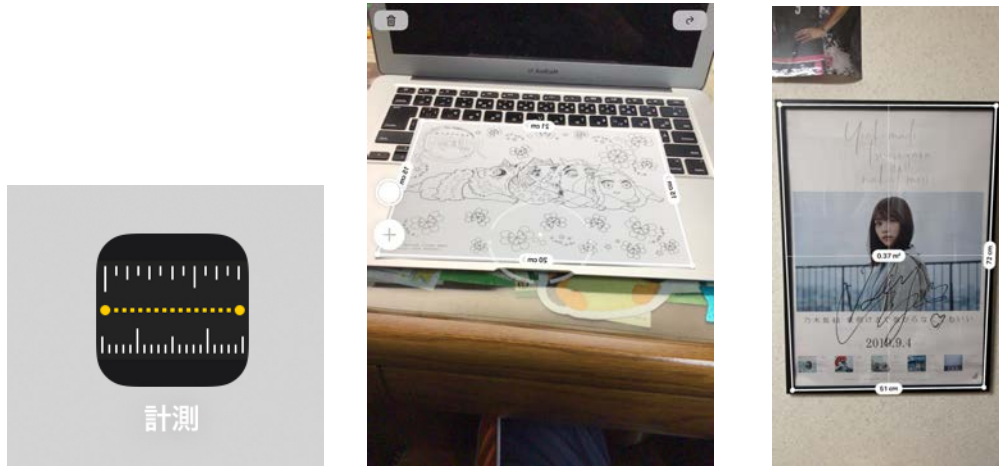


図 1.2 距離や面積を測定できる計測アプリ

Apple 社は、AR アプリを開発するために、ARKit を提供している。ARKit は iPhone, iPad 向けのフレームワークの一つである。ARKit を使うことで、実写映像から平面や物体、画像を認識すること容易になる。また、ARKit は VIO を使うことで、空間認識や自己位置推定を行っている。本研究では、ARKit を使った計測アプリの一つである JustBox を開発した。

本論文では、2 章で、ARKit とそれを使ったプログラムについて説明する、3 章で、開発したアプリ JustBox を説明する。4 章で JustBox の性能を検証し、5 章で結論を述べる。

2章 ARKitの特徴とプログラムの構成

2章ではARとARKitの特徴を説明する。また、本研究で開発するARKitを使ったプログラムの構成や書き方の概要を説明する。

2.1 ARとARKitの位置付け

ARは、実世界にコンピュータで生成した情報を付加することで、実世界を拡張するような技術のことである。AR技術を明確に述べることは難しいが、典型的なARは、カメラで撮影した映像に3DCG画像を違和感なく合成することで、仮想的な物体があたかもそこにある様に見える技術である[1]。

ARと、その類似技術であるVR(Virtual Reality, 仮想現実)やMR(Mixed Reality, 複合現実)は、いずれもコンピュータ技術を利用し、実映像にCG画像を合成させたり、ユーザに没入感を与えたり、映像と人間をインタラクションさせたりすることが可能な技術である。これらの境界を明確に区別することは難しく、Paul Milgramと岸野文郎は、実世界と仮想世界が関係するVRやAR、MRなどの技術は、図2.1のように境界なく連続するVirtual Continuum(VC)を構成していると説明した。この図によると、ARは実世界を主として、それに仮想情報を付与するようなもの、ということである。

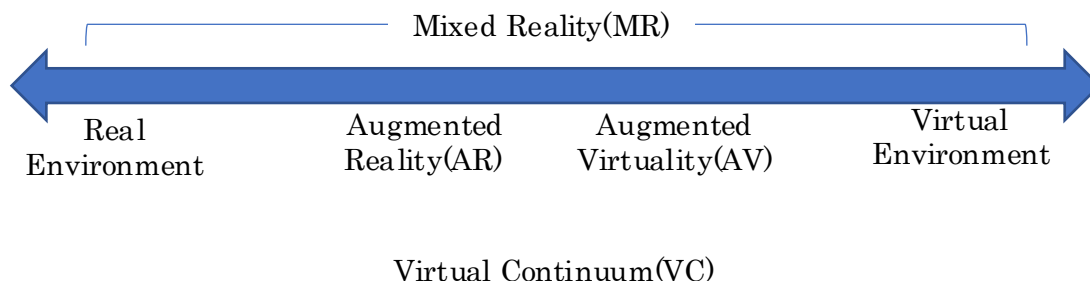


図 2.1 Virtual Continuum(VC)におけるARの位置付け

ARKitは、iPhoneやiPad向けのARアプリの開発を容易にするために、Apple社が提供している開発フレームワークの一つである。ここで開発フレームワークとは、関連するAPIやクラスライブラリをまとめたもののことである。ARKitは2017年のWWDC(Worldwide Developers Conference: 年次開発者会議)で公開され、提供が始まった。その後、毎年バージョンアップが進み、現在はARKit4になっている。ARアプリを開発するには、コンピュータビジョンやコンピュータグラフィックスの技術が必要である。これらをOpenCVやOpenGLなどの汎用性が高いライブラリを使って開発するとARに最適な機能を実現することが容易ではなく、プログラムの量も多くなる。ARKitを利用することで、モーショントラッキング・平面の検出・物体認識・光源推定など、ARに必要な機能を比較的容易に実装することが可能である。

2.2 VIOによる自己位置推定と Point Cloud の取得

ARKitの主要な機能の一つにVIOによる自己位置推定とPoint Cloud（3次元点群）の取得がある。これについて説明する。

スマホのカメラを使って、静止物体や周囲環境を移動しながら動画撮影し、物体上の特徴点をトラッキングすることで、自己位置推定と物体や周囲環境の3D形状復元を行うことができる。この技術をVisual Odometry (VO, カメラを使った空間測定)やStructure from Motion(SfM, 動きからの形状推定)と呼ぶ。しかし、画像情報だけを用いるため、画像特徴が少ない場合には誤動作しやすいという欠点がある。一方、スマホにはジャイロや加速度センサなどの慣性センサが搭載されている。これらのセンサ信号を処理することで、自己位置を推定することができる。この二つの技術を組み合わせ、自己位置と周囲環境の3D推定を高精度化する技術をVisual inertial odometry (VIO)と呼ぶ。ARKitでは、VIOを用いて平面検出などの空間認識や物体の3D形状復元を行っている。

ARKitが行う3D形状復元はPoint Cloudベースのものである。Point Cloudは、3次元空間における点の集まりを意味する。一つの点は3次元座標(x, y, z)で表現する。Point Cloudは立体物の形状を表現する生データの一つである。例えば、図2.2はティーポットの形状をPoint Cloudで表現したものである。

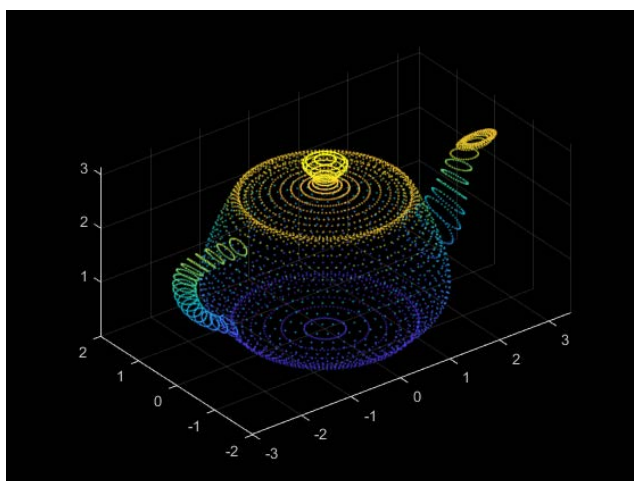


図 2.2 Point Cloud で表現したオブジェクトの例(MathWorks 社の Web サイトより)

ARKitを使うことで、自動的に Point Cloud を取得することが可能になる。ARKit で取得した Point Cloud を描画した例を図 2.3 に示す。Point Cloud は画像中の特徴点をトラッキングすることで計算される。特徴点は、画像中で直線が交差する角や、細かい模様が2次元的に広がっている部分に発生しやすいが、視覚的に濃淡変化が少ない箇所であっても特徴点が発生する場合もある。ただし、その場合には特徴点の密度は低い。

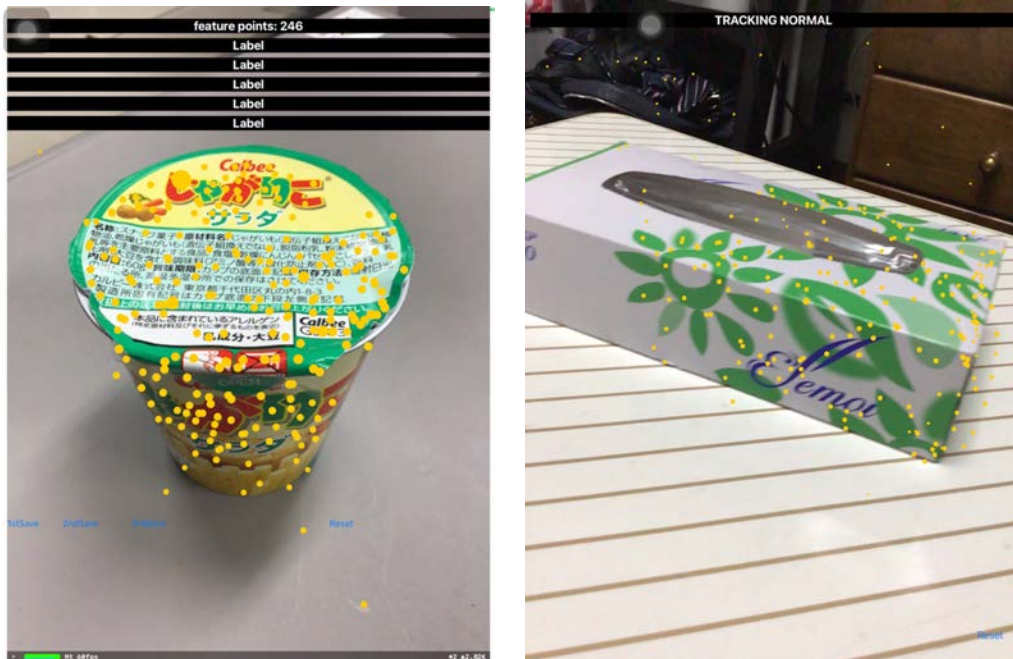


図 2.3 ARKit による Point Cloud の取得

2.3 ARKit を使ったプログラムの構成

本研究では ARKit を使ったプログラムを次の環境で開発した.

開発環境 :

MacBook Air 13-inch Early 2015

MacOS Catalina version 10.15.7

Xcode version 12.3

プログラミング言語 :

Swift

使用端末(iOS デバイス) :

iPad (第7世代) version 14.2

開発の基になるサンプルプログラムとして、参考文献[2]のサンプルコードの一つである“11_FeaturePoints”のプログラムを利用した。このプログラムの動作画面は図 2.3 のような感じで、iPad のカメラで撮影した映像中の特徴点を ARKit の VIO 機能で Point Cloud に変換する。抽出した Point Cloud は、入力画面上に黄色の点として表示される。Point Cloud は iPad のカメラを少し移動させることで発生する。iPad をさらに移動させると、発生した Point Cloud は対応する物体に貼り付いたような感じになり、映像中での物体の移動とともに移動する。対応する特徴点が見えなくなるとその点は消滅する。このように、Point Cloud は特徴点の 3 次元座標を表しており、

iPad を移動させても、その特徴点が明確に見えているかぎり、その 3 次元座標を維持している。

このプログラムにおいて、Point Cloud を抽出するための初期化を行うコードセグメントを図 2.4 に示す。viewDidLoad() は iPad の画面にビューがロードされた時に自動的に呼び出される関数である。図 2.4 では、アプリ起動時の初期化をカスタマイズするためにオーバーライドしている。sceneView は ARSCNView クラスの変数で、ARSCNView クラスは AR コンテンツを表示するためのクラスである。sceneView.session.delegate = self とすることで、ARSessionDelegate プロトコルに準拠したメソッドが呼ばれるようになる。sceneView.scene = SCNScene() でシーン（シーンは描画・表示するもの）で、この段階では描画するものはない）を生成する。let configuration = ARWorldTrackingConfiguration() は ARWorldTrackingConfiguration というクラスを生成している。このクラスを生成することで、デバイス（iPad）の位置と角度を追跡するなどの VIO 機能が利用可能になる。続く 2 行は、水平面検出と光源推定を可能にする設定である。sceneView.debugOptions = [ARSCNDebugOptions.showFeaturePoints] はデバッグオプションの一つを使って、抽出した Point Cloud を点として描画することを設定している。初期化の最後に、sceneView.session.run(configuration) によってセッション（AR 動作を行わせること）を開始している。

```
override func viewDidLoad() {
    super.viewDidLoad()
    sceneView.session.delegate = self
    // シーンを生成して ARSCNView にセット
    sceneView.scene = SCNScene()
    // セッションのコンフィギュレーションを生成
    let configuration = ARWorldTrackingConfiguration()
    configuration.planeDetection = .horizontal
    configuration.isLightEstimationEnabled = true
    // 特徴量を可視化
    sceneView.debugOptions = [ARSCNDebugOptions.showFeaturePoints]
    // セッション開始
    sceneView.session.run(configuration)
}
```

図 2.4 Point Cloud 抽出の初期化を行うコードセグメント

セッションが始まると、ARSessionDelegate プロトコルに準拠したメソッドの中で、実装されているものが、適時、呼び出される。図 2.5 に示す func session(ARSession, didUpdate : ARFrame) はカメラからフレーム（画像）を入力した時に呼び出されるメソッドである。このメソッドの第 2 引数で渡される frame オブジェクトが、最新フレームにおけるさまざまな情報を保持している。guard let pointCloud = frame.rawFeaturePoints で最新フレームから抽出した特徴点に対応する Point Cloud を変数 pointCloud に格納してい

る。pointCloudはARPointCloudクラスのオブジェクトで、表2.1に示すように、そのプロパティとして3次元座標とIDを保持している。

```
func session(_ session: ARSession, didUpdate frame: ARFrame) {
    guard let pointCloud = frame.rawFeaturePoints else {
        return
    }
    statusLabel.text = "feature points: \(pointCloud.__count)"
}
```

図 2.5 カメラから画像が入力された時に呼び出される ARSessionDelegate メソッド

表2.1 ARPointCloudのプロパティ

プロパティ	型	説明
points	[vector_float3]	各特徴点の3次元座標
identifiers	[UInt64]	各点のID

2.4 3D オブジェクトの描画

開発したプログラムで直方体の描画を行う必要があるため、その方法を説明する。直方体などの3D オブジェクトの描画にはSceneKitを用いる。SceneKitは、3Dグラフィックスのためのフレームワークである。

シーンは階層的なシーングラフで表現される。シーングラフのノードは3Dモデルやカメラ、光源に対応する。3次元のシーンを生成するプログラムの流れは次のようになる。

- ① `sceneView.scene = SCNScene()` (図 2.4 の 5 行目) のようにすることで空のシーンを作る。
- ② 3Dモデルやカメラ、光源などシーンに必要な物をオブジェクト化し、それぞれに対応したSCNNodeオブジェクトの該当するプロパティにセットしておく。
- ③ `sceneView.scene.rootNode.addChildNode()`メソッドで、②の全てのSCNNodeオブジェクトをシーンに追加する

直方体を描画する場合について説明する。まず、直方体を表現するSCNBoxオブジェクトを生成する。SCNBoxオブジェクトの大きさや色などのプロパティを設定する。SCNNodeオブジェクト生成し、そのgeometryプロパティにSCNBoxオブジェクトをセットする(②)。このSCNNodeオブジェクトをシーンに追加する(③)。

SCNBoxは、図 2.6 のように、面がすべて長方形で、オプションでエッジとコーナーが丸みを帯びた直方体を定義するクラスである。SCNBoxをオブジェクト化するコードセグメントを図 2.7 に示す。width は物体の幅、height は高さ、length は奥行きを表して

いる。描画するとき、描画角度を設定していないと軸並行な直方体が描画される。中心の座標を設定することで、指定した場所に描画できる。

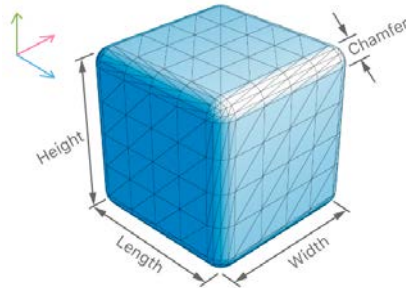


図 2.6 SCNBox のプロパティ

```
let geometry = SCNBox(width: width_x, height: height_y, length: length_z,  
                      chamferRadius: 0.0) //CGFloat型
```

図 2.7 SCNBox をオブジェクト化するコードセグメント

描画した直方体に陰影がないと、直方体であることが分かりづらい。そのような場合には、光源を設定して陰影を表現することができる。図 2.8 に光源を設定するコードセグメントを示す。let lightNode = SCNNode() で新たな光源用の SCNNode オブジェクト（光源ノード）を生成する。let light = SCNLight() で光源オブジェクトを生成する。

lightNode.position = SCNVector3(x: 0, y: 0, z: 0) で光源の位置を設定する。lightNode.light = light で光源ノードの light プロパティに光源オブジェクトを設定する。

scene.rootNode.addChildNode(lightNode) で光源ノードをシーンに加える。

```
// 光源ノードを作り3D仮想空間のルートノードに追加  
let lightNode = SCNNode()  
let light = SCNLight()  
lightNode.position = SCNVector3(x: 0, y: 0, z: 0)  
lightNode.light = light  
scene.rootNode.addChildNode(lightNode)
```

図 2.8 光源を設定するコードセグメント

図 2.9 の左図は光源を設定していない時の画像である。右図は光源を設定し、直方体に陰影を付けた画像である。陰影を付けることで、直方体の形がわかりやすくなっている。



図 2.9 光源の有無の場合の直方体の描画（左が光源なし, 右があり）

これらのプログラミング技術は, 3 章で説明する JustBox の開発において基盤となる技術である. そこに追加の処理や機能を追加し, アプリ開発に取り組んだ.

3章 JustBox の開発

3.1 JustBox について

本研究で開発する JustBox は, カメラで物体を撮影し, その物体をちょうど囲うような箱(直方体)を描くアプリである. 最終的に完成させたアプリの動作画面を図 3.1 に示す. JustBox は「物体を Just (ぴったり) に囲む Box (箱)」という意味である.

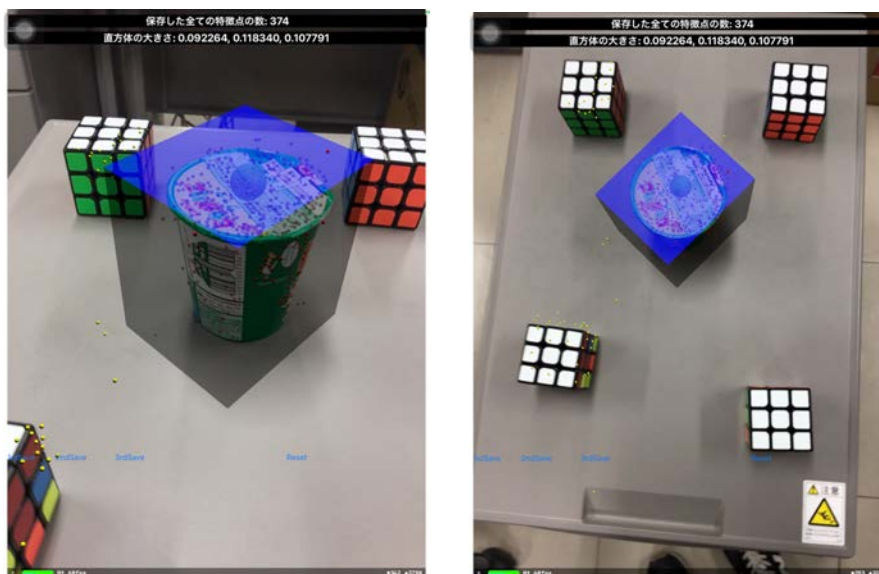


図 3.1 JustBox の動作画面

JustBox は, フリマやコンビニでの個人荷物の発送時などに使うことを想定している. 図 3.2 のように, 店頭で JustBox を使って配送したいもののサイズを計測し, その場でちょうどどの大きさの段ボールを購入できれば, 荷物の配送が楽になる.



図 3.2 Just Box の利用場面のイメージ

3.2 JustBox の処理の流れ

JustBox を使って物体のサイズを測定する処理の流れは次のようになる.

- ① 測定物を机の上に置き, JustBox を起動する. カメラ映像で物体をとらえ, その中心付近の表面をタップする. タップした位置に緑色の点が表示される.
- ② 測定物を周囲から撮影し, 全体を囲む Point Cloud を取得する.

- ③ ②で入力した Point Cloud には，背景など測定物以外の Point Cloud も含まれるので，測定物以外の Point Cloud を除去する．
- ④ ③の Point Cloud を最小の体積で囲む直方体を求める．
- ⑤ 求めた直方体を画面に描画する．

測定物の全体を囲む Point Cloud として，例えば，測定中の各フレームから得る Point Cloud の和集合を，測定物の全体を含む Point Cloud とすることが考えられる．今回の開発では，この考え方を単純化して，ユーザが指定する 3 枚のフレーム(図 3.3)から得る Point Cloud の和集合を，測定物全体を含む Point Cloud とした．

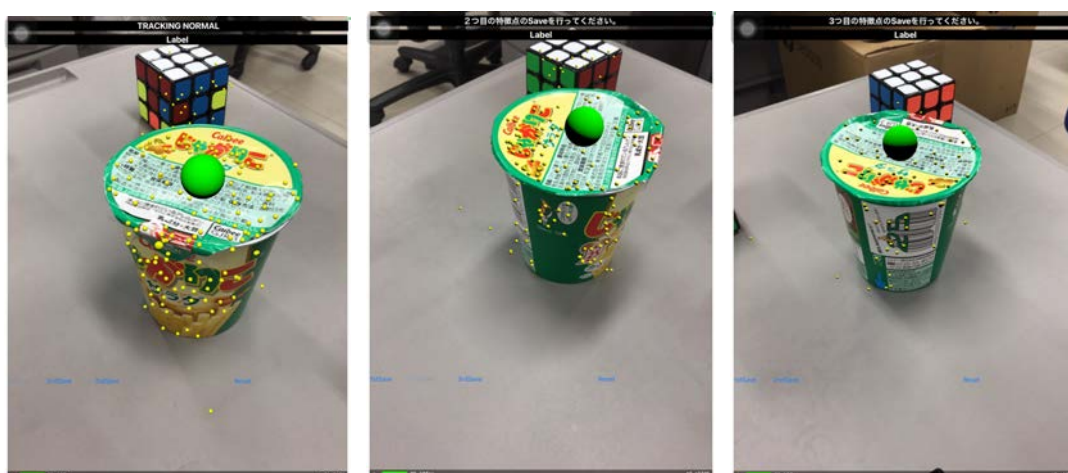


図 3.3 3つの視点から獲得した Point Cloud(左から 1, 2, 3 視点目)

3.2.1 測定物以外の Point Cloud の除去

カメラで撮影した映像には背景や物体の影，物体を置いている机なども映る．図3.3で，中央にある物体(菓子箱)が測定したい物体である場合，後ろに写っているルービックキューブや机の上に発生している Point Cloud はノイズにあたる．取得した Point Cloud の中から，それらをノイズとして除去する必要がある．

このようなノイズを除去する方法として，測定物の表面にある Point Cloud と，そうでない Point Cloud に分ける，ある種の 2 値化が利用できると考えた．2 値化とは，主に画像処理で使われる言葉であり，濃淡画像を白と黒の 2 階調に変換する画像処理技術のことである．図3.4に 2 値化処理の例を示す．

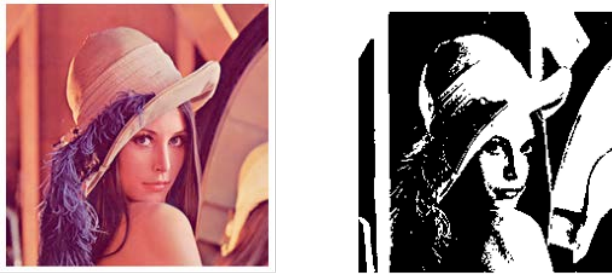


図 3.4 画像処理における 2 値化の例 (右が 2 値化処理後)

2 値化では, あらかじめ閾値を設定し, 画素値が閾値より大きければ白, 小さければ黒に変換する. その閾値の決定方法として判別分析法がある. 判別分析法は, 2 クラス(白と黒)の分布における分離度が最大になるように閾値を決める方法である.

判別分析法の閾値の計算法を説明する. 数値の集合 (画像の 2 値化の場合は画素値) を閾値 t で 2 クラスに分け, それぞれの分散 (σ_1^2, σ_2^2), 平均値 (ω_1, ω_2), 要素数 (m_1, m_2) を求める. そこからクラス内分散 σ_w^2 , クラス間分散 σ_b^2 を求め, 分離度 S を求める. 分離度 S はクラス間分散とクラス内分散の比で定義され
$$\frac{\sigma_b^2}{\sigma_w^2} = \frac{\sigma_b^2}{\sigma_t^2 - \sigma_b^2}$$
 である. ここで, 全分散 σ_t^2 は閾値に関係なく一定なので, クラス間分散 σ_b^2 が最大となる閾値を求めればよい. さらにクラス間分散の式の分母も, 閾値に関係なく一定なので, クラス間分散の分子 $\omega_1 \omega_2 (m_1 - m_2)^2$ が最大となる時の閾値 t を求めるだけで良い. 濃淡画像の 2 値化では画素値は 0 から 255 の整数であるから, 256 通りの閾値を設定し, 分離度が最大になるものを選ぶ.

判別分析法を Point Cloud のノイズ除去に用いる方法を検討した. 考え方とし, 取得した Point Cloud の各点に一つの評価値を与え, その評価値集合を判別分析法で 2 クラスに分ける, ということである. その評価値として, 測定物の中心位置から各点までの距離とすることが妥当と考えられる. 測定物の中心位置は不明であるから, 今回は, 中心に代わる代表点をユーザが画面上でタップすることとした. 本研究における判別分析法による閾値の決定方は以下ようになる.

- ① 測定物表面の代表点をユーザがタップする.
- ② 取得した Point Cloud の全ての点と代表点のユークリッド距離を求める.
- ③ ②で得たユークリッド距離の集合を判別分析法で 2 クラスに分ける.

ここで, ③の処理において, 可能性のある全ての閾値を試す必要があるので, 測定した全ての距離値を使った.

図3.5に、本実験で閾値を決定した際のヒストグラムの例を示す。このように、取得するPoint Cloudを閾値 t で、物体に近い点(表面上の点)と物体から離れている点に分けることができる。

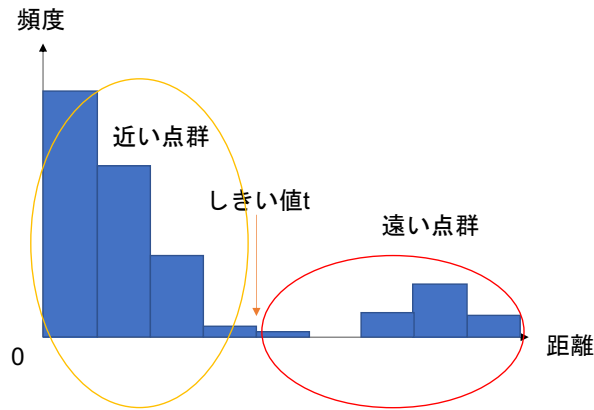


図 3.5 Point Cloud を距離のヒストグラムで2クラス化できる例

3.2.2 Point Cloud を囲む直方体の求め方

次に、ぴったりの箱(直方体)を描画するために、最小の直方体を求める方法を説明する。Point Cloudの中から、 X 座標値が最大と最小の点、 Y 座標値が最大と最小の点、 Z 座標値が最大と最小の点の6点を抽出する。これら6つの最大値、最小値を X_{max} , X_{min} , Y_{max} , Y_{min} , Z_{max} , Z_{min} とする。図 3.6 に描画される直方体のイメージ図を示す。

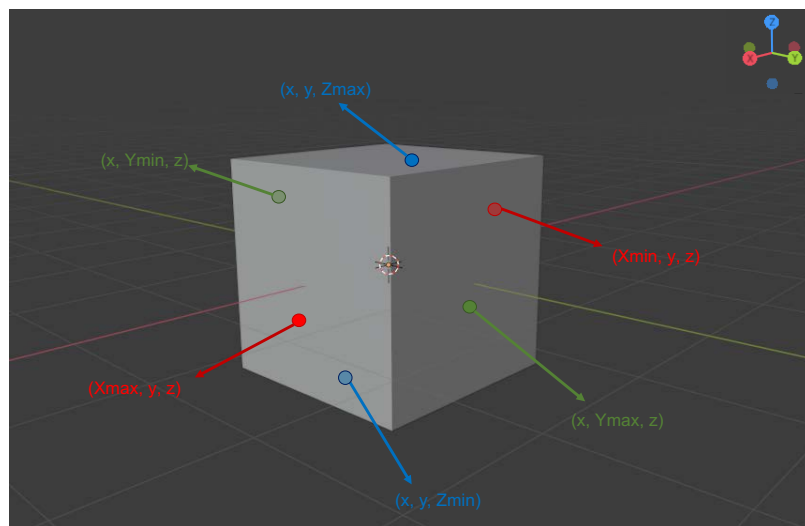


図 3.6 最小の直方体で Point Cloud を囲むイメージ

これら6つの値を使って、直方体の描画に必要な width(物体の幅), height(高さ), length(奥行き)と、直方体の中心位置 (X, Y, Z) を次のように求める。

$$\text{width} = X_{max} - X_{min}$$

$$\text{height} = Y_{\max} - Y_{\min}$$

$$\text{length} = Z_{\max} - Z_{\min}$$

$$X = (X_{\max} + X_{\min})/2$$

$$Y = (Y_{\max} + Y_{\min})/2$$

$$Z = (Z_{\max} + Z_{\min})/2$$

4 章 実験結果

本章では Just Box の動作検証を行った。また、見つかった課題について記述する。

4.1 アプリの動作検証

開発したアプリを 3.2 節で説明したように動作させた。図 4.1 の菓子箱(じゃがりこ)を測定物体として用い、この物体をちょうど囲う直方体を描画した。この物体は幅 8.6cm、奥行き 8.6cm、高さ 8.8cm である。



図 4.1 計測した物体

アプリを手順通りに実行し、描画された青色の直方体を図 4.2 に示す。この時、2 値化の結果を分かりやすく確認するために、周りにルービックキューブを並べ、ノイズとなる Point Cloud を検出させた。2 クラスに分離した後の Point Cloud において、近い点は赤色の丸、遠い点は黄色い丸で描画した。

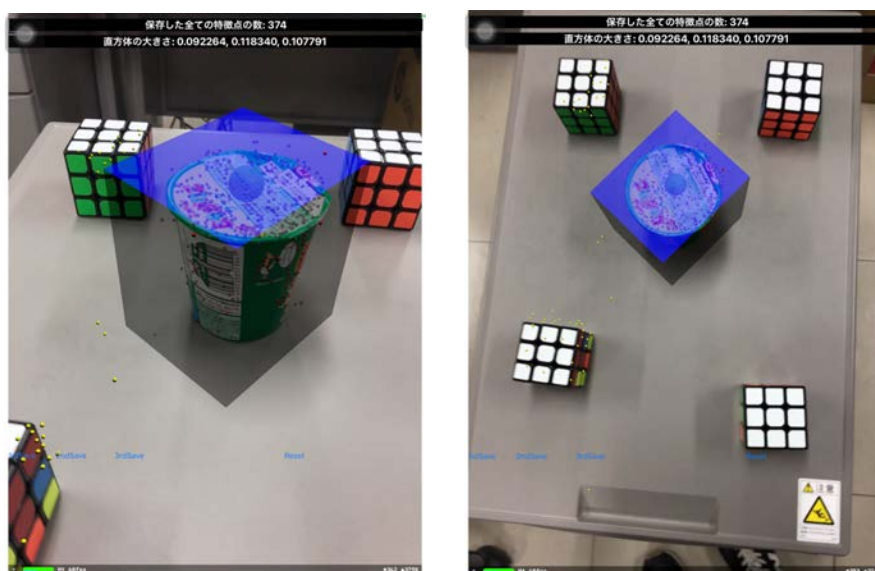


図 4.2 じゃがりこの菓子箱を測定した時の画像

図 4.2 では、描画されている青色の直方体が、ちょうど物体を覆っているように見えるので、見た目は Just に近い形で描画できていることがわかる。描画した直方体の大

きさは、幅 9.2cm, 奥行き 10.8cm, 高さ 11.8cm であった. 実際の物体寸法との誤差は、幅 +0.6cm、奥行き+2.2cm、高さ+3.0cm である.

図 4.3 に、取得した Point Cloud について、距離のヒストグラムを示す. 近い側の点と遠い側の点を分ける閾値 10.81cm で、図のように 2つのクラスに分けられていた.

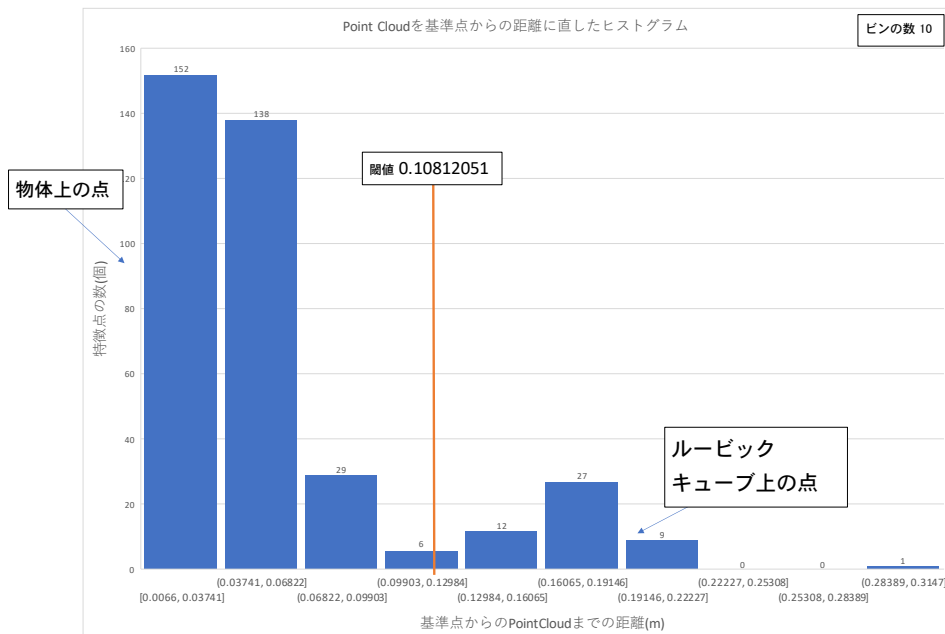


図 4.3 取得した Point Cloud の距離のヒストグラムと閾値

図 4.2 を見る限り、測定物の表面にある Point Cloud はすべて赤色の点で表されている. また、ルービックキューブ上の Point Cloud や机の上の Point Cloud が黄色の点で表されている. これらのことから、実装した判別分析法などのアルゴリズムなどは正しく動作し、それにも関わらず、上記の誤差が発生したものと考えられる.

4.2 サンプルを増やした検証

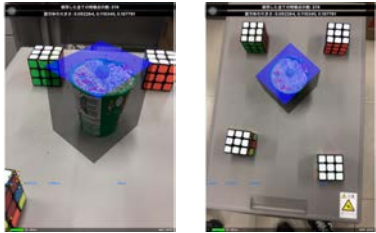

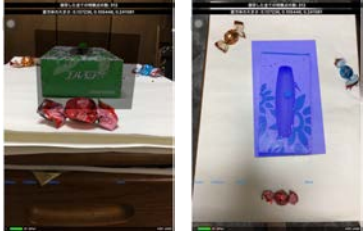
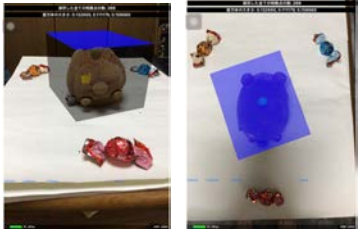
測定物のサンプルを増やして計測を行い、アプリの精度を確認した. 形の違う物を測った時の精度や、その時見つかった課題について調査した. 追加して測定した 3つの物体を図 4.4 に示す.



図 4.4 追加で測定した物体の外観

左からテニスボール（球），ティッシュボックス（直方体），ぬいぐるみ（不定形）である．これらに，4.1 で使用した円柱の菓子箱も加えて検証する．測定物の実際の大きさと，測定された大きさ，描画時の写真を表 4.1 にまとめた．

表 4.1 測定物と測定結果の一覧

測定物名	実際の大きさ	測定された大きさ	描画された直方体
(例)	幅, 奥行き, 高さ	幅, 奥行き, 高さと誤差	
菓子箱 (じゃがりこ) 円柱	8.6cm 8.6cm 8.8cm	9.2cm (+0.6cm) 10.8cm (+2.2cm) 11.8cm (+3.0cm)	
テニス ボール 球	7.0cm 7.0cm 7.0cm	7.4cm (+0.4cm) 7.8cm (+0.8cm) 7.7cm (+0.7cm)	
ティッシュ ボックス 直方体	23cm 11.5cm 5.5cm	24.1cm (+1.1cm) 13.7cm (+2.2cm) 10.5cm(+5.0cm)	
ぬいぐるみ 不定形	11.5cm 10.0cm 15.0cm	13.2cm (+1.7cm) 11.1cm (+1.1cm) 16.0cm (+1.0cm)	

結果として，全ての場合で，物体を囲う直方体の描画に成功している．特に，テニスボールに関しては，まさに Just な直方体で覆われていると言える．一方，菓子箱の高さで +3.0cm, ティッシュボックスの高さで +5.0cm の差がでた．これらは，Point Cloud を取

得するためにカメラを動かした時に、残像的な現象で不要な Point Cloud を測定してしまったものと考えている。これらの点はノイズ除去でも除けない距離の点であった。

4.3 課題

アプリ検証の過程で見つかった課題として、Point Cloud を 2 クラス化するときの失敗と、Point Cloud を最小体積で囲む直方体が軸平行でない場合の問題がある。

4.3.1 Point Cloud の 2 クラス化に失敗する場合

図 4.5 に、Point Cloud を 2 クラスに分割することに失敗した場合を示す。この例では、菓子箱の Point Cloud が上と下に分けられている。最初を取得した Point Cloud が菓子箱のものだけであったことが理由と考えられる。現在の 2 クラス化のアルゴリズムは、全ての場合で正しく動作するわけでない。極端な場合として、最初を取得した Point Cloud が分割すべきでないような場合には、図 4.5 の結果になる。



図 4.5 Point Cloud を 2 クラスに分割することに失敗した場合

4.3.2 体積最小の直方体が軸平行でない場合

図 4.6 に、囲いたい物体に対して、推定された直方体が斜めに描画された場合を示す。このような直方体は JustBox とは言えない。つまり、Point Cloud を体積最小で囲む直方体ではない。3.2.2 で説明した方法で直方体を求めると、直方体の辺が 3 次元座標の軸と平行な軸平行な直方体になる。これは、必ずしも Point Cloud を囲む体積最小の直方体ではない。

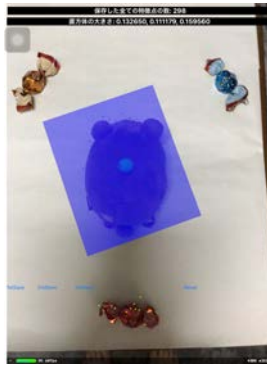


図 4.6 推定した直方体が物体に対して斜めになる場合

この問題は、Point Cloud の 3 次元座標集合を主成分分析することで解決できると考えている。主成分分析を用いて、最小体積の長方形を求める技術の概要を以下に示すが、本研究ではアプリでの開発は行わず、残った課題とする。

主成分分析は、多次元データの要約などに使われる解析手法である。データの要約ができると、データの特徴を判断しやすくなる。今回の問題に適用すると、Point Cloud が表す 3 次元座標集合のデータを主成分分析する。主成分分析の具体的な計算は、分散共分散行列を固有値分解し、最も大きい固有値に対応する固有ベクトルを第 1 主成分、次に大きい固有値に対する固有ベクトルを第 2 主成分、残った固有ベクトルを第 3 主成分とすることである。このようにして求めた 3 つの主成分方向を 3 軸として、3.2.2 の方法で直方体を求めれば、Point Cloud を囲む体積最小の直方体になっているはずである。

5 章 結論

本研究では ARKit の VIO 機能を利用したアプリ Just Box を開発した。JustBox はカメラで撮影した物体を Just に収める直方体を計算し、画面に表示するアプリである。このアプリは、ARKit の機能を使って Point Cloud を取得し、本論文で提案した手法で測定物体表面の Point Cloud を抽出する。そして、その Point Cloud を囲む最小の直方体を求める。

JustBox で 4 種類の測定物を測定し、これらの物体を Just に収める直方体を描画することができた。しかし、Point Cloud をうまく取得することができず、Just な直方体を計算することに失敗することもあった。その理由の一つは、カメラを動かした際に、消えたり増えたりする Point Cloud が残像として反応することであった。また、別な問題として、最初に取得した Point Cloud から測定物体表面の Point Cloud を抽出する処理が正しく動作しない場合もあった。さらに、現在の直方体の大きさの計算方法は、不十分であることがわかった。主成分分析を使うことで、最小体積の直方体を求めることも残った課題である。

最後に、本研究によって、ARKit の VIO 機能を用いることで、スマホ画面に表示される空間の中で自己位置の推定が行われ、目の前の物体を簡単に計測できることがわかった。論文では、VIO と Point Cloud に焦点を当てて考えてきたが、ARKit にはたくさんの機能があり、それらはアプリ開発において有効である。

参考文献

- [1] Erin Pangilinan, Steve Lukas and Vasanth Mohan, “Creating Augmented and Virtual Realities: Theory and Practice for Next-Generation Spatial Computing,” O’ Reilly Media, 2019.
- [2] 堤修一 , 「実践 ARKit 」 , 電子書籍 v1.3 (PDF). 及び, そのサンプルコード集, <https://github.com/shu223/ARKitBook>.

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

[JustBox]

Xcode で作成した自作アプリ「JustBox」である.

11_FeaturePoints のサンプルコードを改良し作ったプログラムである.

判別分析法の計算や, 直方体の描画などのコードは viewController.swift にある.