コンピュータ理工学特別研究報告書

題目

ARKitの認識機能に関する調査と
ARアプリの開発

学生証番号　544863

氏名　田中誠也

提出日　令和2年1月22日

指導教員　蚊野　浩

京都産業大学
コンピュータ理工学部

要約

AR（Augumented Reality, 拡張現実）は実世界を仮想世界の情報によって補強する技術である．その中でも，カメラで撮影した実写映像に３次元 CG 画像を違和感なく合成する AR が一般的である．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 ARKit がある．本論文では ARKit の認識機能について調査し，これを用いたデモアプリを開発した．

ARKit は iOS フレームワークの一つで，これを使うことで AR のプログラム開発が容易になる．ARKit の認識機能として，平面認識，画像認識，物体認識がある．平面認識はシーン中の水平面あるいは垂直面を認識する機能である．画像上の特徴点が極めて少ない場合や，少ない特徴点が直線的に並ぶ場合を除いて，平面を認識する．ただし，厳密に平面性を検証しているわけでない．認識した平面に対して VIO（Visual-inertia Odometry）を用いることで，３次元 CG 画像を違和感なく合成することが可能である．画像認識は，カメラで撮影したシーンの中で，登録した画像を認識する機能である．画像の拡大縮小，回転による変形だけでなく，斜めから見た場合の射影歪みや被写体の一部が隠れた場合でも認識することができる．物体認識は立体物を認識する機能である．事前に立体物をスキャナアプリで測定することにより，認識可能になる．物体認識の場合は，360° あらゆる方向から認識できる．

次に，ARKit を使ったデモアプリを２つ作成した．第一のアプリは「物体認識による移動物体のコース制御」であり，決められたコースを走る移動物体（車）が障害物によって走るコースを変えるというものである．第二のアプリは「画像認識による販売促進」で，画像認識の結果を使って EC サイトにリンクしたり広告を出したりするものである．

結論として，ARKit は短いソースコードで高度な認識機能を実現することができ，AR アプリを作成するためのプラットフォームとして優れている．一方，内部の詳細がブラックボックスであるため，動作を完全に理解することが難しいという側面もある．しかし，AR デモや AR プロジェクトを短時間で実現できることが利点である．

目次

1章 序論

Virtual Reality(VR,仮想現実)技術は，2015年前後にOculus Rift，ソニーのPlay Station VR，台湾HTCのViveなどのHead Mount Display(HMD,ヘッドマウントディプレイ)が商品化され，手にする機会が増えている．これ以降も活発に研究開発されており，多くのゲームやアプリケーションが商品化されている．一方，その頃のAugmented Reality (AR,拡張現実)技術は研究段階に止まっており，商用化は活発でなかった．しかし GPS 情報をもとにスマートフォン越しにポケモンがその場所にいるかのように表現できるARアプリ Pokémon Go が世界中で大ヒットし，社会現象になった．これ以降，ARの認知が高まり，実用化の例が増えている．例えば，IKEAの家具を自分の部屋に配置して収まり具合を確認できる「IKEA Place」などのアプリがある（図1.1）．

図1.1 IKEA Placeのプロモーションビデオの１シーン

(https://www.youtube.com/watch?time_continue=54&v=va41WiGJPt8&feature=emb_logo)

アニメーション「電脳コイル」ではメガネ型のARデバイス「電脳メガネ」が描かれている．これは図1.2のように，メガネ越しに見ている現実世界に電脳世界の情報を重ねて表示するデバイスである．電脳メガネに類する現実のデバイスとして，Google glassやマイクロソフトのHololensがある．

図1.2 電脳コイルのワンシーン（バンダイチャンネルより引用）[1]

スマホ型であれメガネ型であれ，AR を実現するには，現実世界の情報をコンピュータに認識させる必要がある．そのための仕組みとして，マーカー型，マーカーレス型，GPS 情報を使う位置情報型，空間認識型がある．マーカー型はあらかじめ形や輪郭が決まっている画像を登録しそれを認識させる方法である．マーカーレス型は任意の画像を認識させる方法である．空間認識型は立体物を空間的に認識するため，あらゆる視点から認識が可能である．本研究で調査・研究する ARKit はこれらすべての機能を持ち合わせている．

ARKit は Apple から提供されている iOS フレームワークの一つである．これを使うことで，AR アプリの開発が比較的容易になる．なお，Google からも ARKit と同様の機能を持つ AR Core が提供されている．ARKit は実写画像中の平面認識，画像認識，物体認識が可能である．ARKit は visual-inertial odometry を基礎としている．Visual-inertial odometry とはカメラ画像と慣性センサを用いて端末デバイスの位置を推定する技術である．これらを使って複数ユーザ間で AR 空間を共有することなどが可能になっている．

ARKit の使い方はドキュメントで説明されてはいるが，実際，どこまで平面認識が可能であるかは不明である．例えば，ARKit のドキュメントに“its analysis of captured video images detects an area that appears to be a flat surface.”（平らだと思われる領域を検出する）とあるが，“flat surface”とはどのような場合か，など明確でない．そこで，本研究では ARKit の主要な 3 つの認識機能について詳細に調べた．以下，２章で AR と ARKit について説明する．3 章で ARKit の平面認識，画像認識，物体認識の３つの認識機能について詳細に説明する．４章で ARKit を使った AR アプリのデモを説明し，５章で結論を述べる．

2 章 Augmented Reality と ARKit

実写映像や CG 画像をディスプレイで観察することにとどまらず，コンピュータ技術を利用することで，没入感を与えたり，実写映像と CG 画像を違和感なく合成したり，映像と人間をインタラクションさせたり，などが可能になる．これらの技術を Virtual Reality(VR,仮想現実)や Augmented Reality(AR,拡張現実)，Mixed Reality(MR，複合現実)などとよぶ．Paul Milgram と岸野文郎は，これらの技術は図 2.1 のように，境界なく連続した一連の技術であり，全体として Virtual Continuum(VC)を構成するとしている[5]．２章では，本研究で扱う AR と ARKit について説明する．

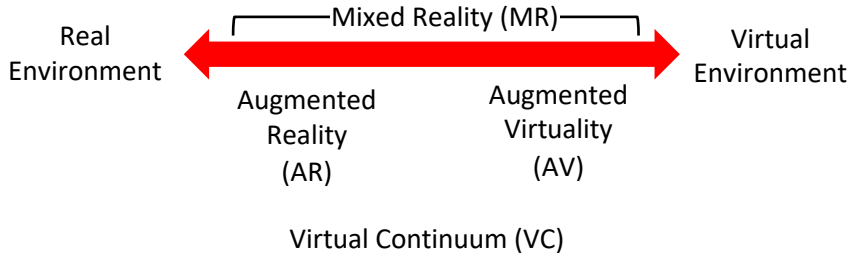


図 2.1 Virtual Continuum における AR の位置付け

2.1 本研究で扱う AR

図 2.1 が示すように，AR は現実環境が主たるもので，それをコンピュータが生成する仮想環境によって補強するものである．扱う環境によって，様々な AR を考えることができる．本研究では，実写映像に CG 画像を違和感なく合成する，というタイプの AR について論じ，最後に AR アプリのデモプログラムを作成する．

本研究で扱う AR は，実写映像を処理することと，実写映像に CG 画像を重畳表示することが必要になる．これを行うための基本的なツールとして OpenCV と OpenGL がある．しかし，これらは AR 用のツールとして開発されたものではないため，AR アプリの開発に用いる場合，多くの AR 機能を自作しなければならない．そこで実際的には，AR 開発用のフレームワークを利用することが多い．AR フレームワークの代表的なものは Apple の ARKit と Google の ARCore である．ARKit は iPhone などの iOS デバイスのアプリを開発するためのフレームワーク，ARCore は Android デバイスのアプリを開発するためのフレームワークである．これらのフレームワークは実写映像とインタラクションするための一群の API とツールである．ARKit と ARCore 以外に，マイクロソフトが Hololens 用に準備したツールなどがある．本研究では ARKit を用いた．

2.2 ARKit

ARKit は iOS で動作するフレームワークである．2017 年の WWDC（Worldwide Developers Conference：年次開発者会議）で公開され，Apple から提供が始まった．そして 2018 年に ARKit2 としてアップデート，2019 年に ARKit3 としてアップデートされている．本研究では ARKit2 までの機能を扱う．

ARKit の特徴の一つは AR 空間を複数のユーザで共有できることである．図 2.2 にそのイメージ図を示す．この図は，二人のユーザが同じ実空間（図の場合，テーブル）を観察しており，そこに同じ仮想空間（図の場合，テーブルの上に乗っている積み木が仮想空間で，CG 画像として生成している）が再現できていることを示す．これを実現している要素技術の一つが visual-inertia odometry(VIO)である．VIO については 2.3 節で詳しく説明する．

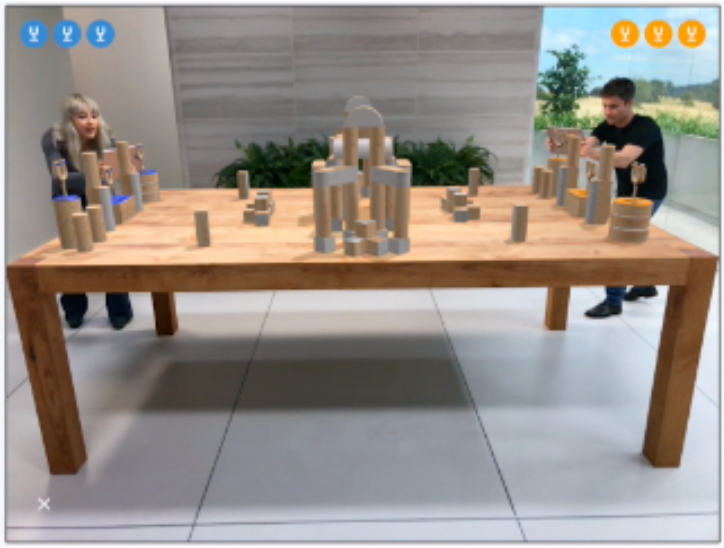


図 2.2 二人のユーザが AR 空間を共有しているイメージ図
(https://www.apple.com/jp/newsroom/2018/06/apple-unveils-arkit-2/)

ARKit の主な機能を簡単に説明する．

-ARKit1-

- ワールドトラッキング
  - 実写映像中の平面を検出し，その平面に対する iOS デバイスの位置と向きを計算することで，実写映像をトラッキングしながら仮想環境（CG 画像）を違和感なく重畳表示することを可能にしている．違和感のない重畳表示とは，iOS デバイスの視点移動に伴う実写映像の変化に応じて，CG 画像の視点が同様に変化することなどを意味している．
- 画像認識

- 実写画像中の平面物体を画像として認識し，その位置と向きをトラッキングする．

-ARKit2 以降-

- 物体認識
  - あらかじめ，立体スキャナアプリ（立体物の形状をスキャンする iOS デバイス用のアプリ）で対象物をスキャンすることで，その物体を認識できる．
- フェイストラッキング
  - 実写映像中の顔を検出し，それに合わせた仮想コンテンツを重ねることで，顔の表情をリアルタイムでアニメーション表示する．

-ARKit3 以降-

- モーショントラッキング
  - 実写映像の人物をトラッキングし，その人物と同じ動きを仮想キャラクターに適用することで，人物の動きを可視化する．
- その他の機能
  - レンダリングエフェクトとして，カメラで撮影した環境プローブ画像を使って，それが写り込んだ仮想オブジェクトをレンダリングする．
  - マルチユーザ機能として，ほかのデバイスと通信して共有 AR 空間を生み出す．

などがある．

本研究で扱うのはワールドトラッキングのための平面認識と画像認識，物体認識である．その他の機能については，ここで若干，説明を追加する．

フェイストラッキングは顔を認識し追跡するものである．これを高精度に行うことができ，顔に CG 画像を重ね合わすことができる．iPhone X 以降の端末で利用できる「アニ文字」（図 2.3，自分の声と表情に合わせて動くアニメキャラクタ）はこの機能を使っていると思われる．

図 2.3 アニ文字の例

（https://support.apple.com/ja-jp/HT208190）

モーショントラッキングはリアルタイムで体全体をトラッキングできる．さらにオク

ルージョン機能が追加されており，CG 画像と人の重なりを正しく表現できる．ただし，iPhoneX 以降の端末にある TrueDepth カメラが必要である．

2.3 Visual-inertial odometry と Point cloud

Odometry は odometer（走行距離計）の関連語で，一般的には，移動ロボットの移動量を計測する手法である．最も基本的な方法は wheel odometry で，車輪の回転数と車輪の切れ角から移動量を計算する．しかし，この方法は車輪が滑るなどして空転した分も計算するなど誤差が大きい．

これに対して，移動ロボットに搭載したカメラで環境を撮影した画像から自己位置推定を行う方法が visual odometry(VO)である．Visual odometry の原理は，コンピュータビジョンの分野で Shape from Motion(SfM，動きからの形状推定)と呼ばれている技術である．シーン中の特徴点を異なる視点から撮影した画像を用いて幾何学的な計算を行うことで，特徴点の３次元座標を復元することができる．SfM の欠点として，視点位置が大きく移動する場合には特徴点の対応づけが難しくなり，移動量が少ない場合には３次元復元の精度が悪い，ということがある．

Visual odometry を発展させたものが visual-inertial odometry(以後 VIO)である．Apple のドキュメントには「iOS デバイスのモーションセンサから得られる情報を，デバイスのカメラで捉えたシーンのコンピュータビジョン解析と組み合わせる．ARKit は，シーン画像に含まれている注目すべきフィーチャ（特徴点）を認識し，それらのフィーチャについて，ビデオフレーム間での位置の違いをトラッキングし，その情報をモーションセンサのデータと比較する．結果として得られるのは，デバイスの位置とモーションに関する高精度のモデルである．」[2]とある．つまり，VIO は VO にモーションセンサで得た情報を加えて精度を高めたものである．

３次元空間における点の集まりを point cloud（点群）とよぶ(図 2.4)．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り，visual odometry は point cloud を利用している．

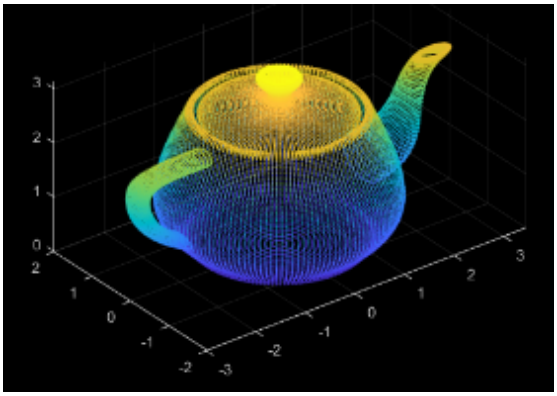


図 2.4 Point cloud で表現した立体物の例（MathWorks 社の Web サイトの例）

ARKit で取得した point cloud の例を図 2.5 に示す．一般に visual odometry は画像中のコーナー的な特徴を利用して幾何学的な計算を行うので，ARKit で取得する point cloud は画像中のコーナー特徴点になる可能性が大きい．そして，本研究で検証する平面検出や物体認識，画像認識は point cloud のデータを利用している可能性が高い．

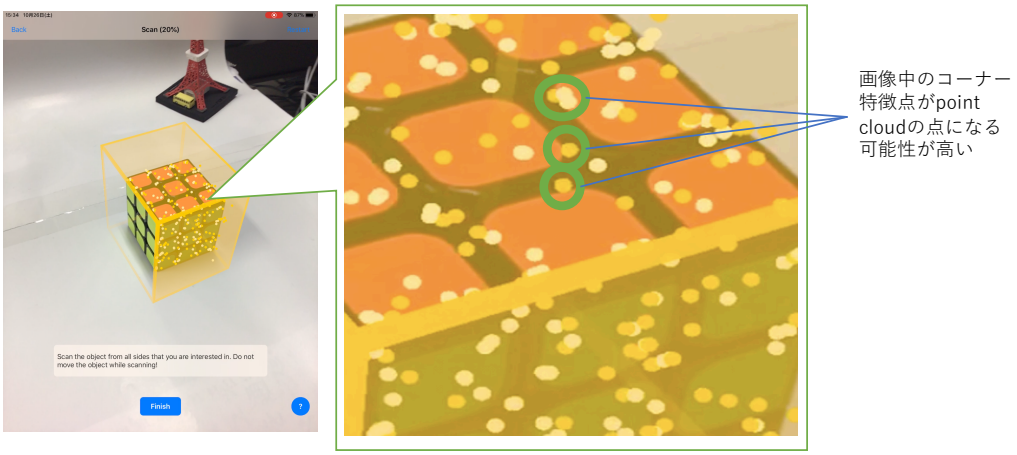


図 2.5 ARKit における特徴点の取得

3 章 ARKit の認識機能の検証

３章では，ARKit を利用して AR を実装する上で基本的な機能になる平面認識，画像認識，物体認識を検証する．本実験は以下の開発環境で実施した．

開発環境：

MacBook Air 11-inch early2015

MacOS Catalina version 10.15

Xcode version11.1

プログラミング言語：

Swift

使用端末（iOS デバイス）：

iPad(第 6 世代)　version 13.1.3

3.1 平面認識

3.1.1 平面認識の理論

映像中から平面を認識するための，一般的なアルゴリズムは次のようなものである．

映像から平面を認識するアルゴリズム：

① 視点を移動して撮影した複数の画像からコーナー特徴を抽出し，それらを画像間で対応させ，さらに，SfM 理論に基づく幾何学的な計算を行うことで point cloud を得る．

② Point cloud から，一つの３次元平面上に存在すると推定される部分集合を抽出し，それを平面の point cloud とする．

③ 平面の point cloud に平面フィッティングを行い，平面の方程式を求める．

④ Point cloud が分布する領域を用いて，平面の境界を長方形や多角形などで近似する．

以上．

ARKit は水平面の認識と垂直面の認識が可能であり，それらのための API が用意されている．平面認識のアルゴリズムの詳細は不明なので，実験的にその機能を明らかにする．以下では主に，水平面の認識機能について報告する．

3.1.2 平面認識の実験

ARKit を使った平面認識のための Swift によるコードセグメントを図 3.1 に示す．

```
1       let configuration = ARWorldTrackingConfiguration()
2       configuration.planeDetection = [.horizontal]
3       sceneView.session.run(configuration)
```

図3.1 平面認識を行うためのコードセグメント

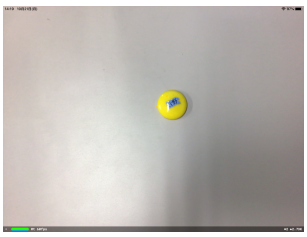
１行目で ARWorldTrackingConfiguration クラスのインスタンスを生成し，変数 configuration に代入している．ARWorldTrackingConfiguration は iOS デバイスの位置と方向を計算し,設定するためのクラスであり,そのための対象物として平面と画像,物体（立体物）を利用することができる．水平面認識を行う場合，２行目のように，メンバ変数 planeDetection に“[.horizontal]”をリテラルとして代入する．垂直面認識を行う場合，“[.vertical] ”を代入する．両方を同時に認識させる場合，“[.horizontal,.vertical] ”を代入すればよい．(複数の要素を同時に設定する場合には[ ]が必要であるが，単独であれば，[ ]は不要である．)３行目で，予め用意した sceneView クラスに平面を検出するようにした configuration を設定して，実行させることで平面検出が可能になる．sceneView は SceneKit と ARKit で AR コンテンツを表示,管理するものである.基本的に SceneKit が扱うものなので本論文では説明しない.

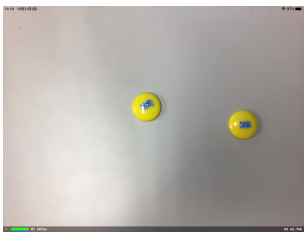
予備実験として図3.2左の特徴のない白い面と，図3.2中央の特徴の多い面を水平面認識させた．使用するプログラムは図3.1のコードセグメントに加えて，検出した水平面を青い四角形で覆うように表示させるコードを書き加えたものである．その結果,特徴の無い面は認識に失敗し，特徴の多い面は図3.2右のように認識に成功した.

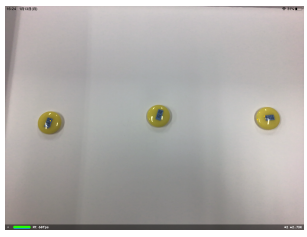
図3.2　特徴の無い面(左)，特徴の多い面(中央)，検出された面（右）

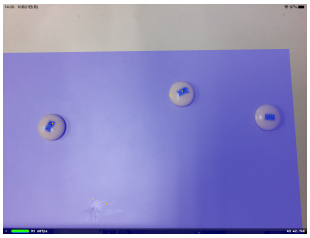
平面認識機能は特徴の多い面ほど認識しやすく,特徴の少ない面ほど認識されにくい.そこで特徴の無い画像から少しずつ特徴量を増やし，また，特徴の配置をコントロールして，水平面の認識の可否を確認した．その結果を図 3.3 に示す．図 3.3 で領域を青色で表示しているものは成功した場合,表示していないものは失敗した場合である.図 3.3 の(a)，(b)，(c)が示すように，マグネットが２つ以下の場合と，３つのマグネットが直線的に存在する場合は平面の認識に失敗している.特徴の分布が２次元的に広がる場合や,ほぼ直線的であっても,特徴点が多くなれば平面の認識に成功する.この実験から,ARKit の平面認識機能は，特徴の数が極端に少ない場合や，少ない特徴が直線的に存在

する場合を除いて，成功することがわかった．


(a) 失敗（マグネット１つ）


(b) 失敗（マグネット２つ）


(c) 失敗（直線上にマグネット３つ）


(d) 成功（一般の位置にマグネット３つ）

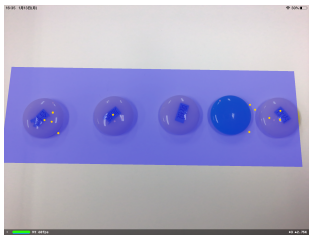

(d) 成功（直線上にマグネット５つ）

図 3.3 画像中のマグネットの個数・配置と検出の可否

複数の平面が認識されると図 3.4 のような形になる．この例では，認識した全ての平面を表示させた．図 3.4 の右側で，机・椅子・その他の小物が混在する領域を平面として認識している．このことから ARKit は厳密に平面性を判定しているのではないことがわかる．

図 3.4 複数の平面が検出された場合

続いて，VIOに基づくワールドトラッキングの機能を検証した．実写画像にCG画像を重畳した状態でiOSデバイスを移動させたとき，ARKitを用いることで，CG画像も実写画像の一部であるかのように見えかたが変化する．ARKitではこれをVIOで実現している．一方，カメラ映像の見え方だけについて言えば，カメラを移動させた場合とカメラを固定しシーンを移動させた場合で，区別がつかないことがある．つまり，平面

を認識した状態で，図3.5左で示すようにiOSデバイスを反時計回りに90°回転させた場合と，図3.5右で示すようにiOSデバイスを固定し平面を時計回りに90°回転させた場合は，平面の見え方に違いがない．そこで，この２つの場合で重畳したCG画像の見え方がどのように変わるかを検証した．

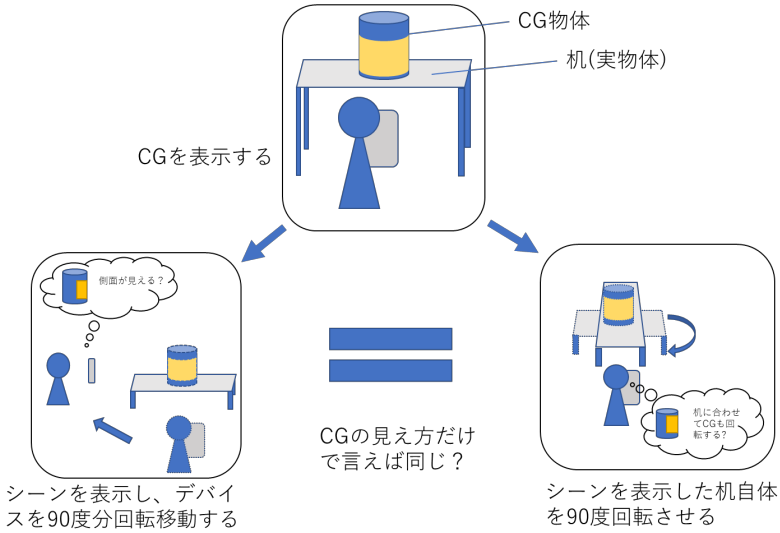


図3.5 端末を持って回転する場合(左)と，実世界の平面を回転させる場合(右)

ウイスキボトルの３次元 CG 画像を使って実験したところ，端末を持って 90° 回転移動した場合，図 3.6 のように重畳した CG 映像も正しく回転した．一方，端末位置を固定して平面を逆方向に 90° 回転させた場合には，CG 画像にほとんど変化がなかった（図 3.7）．VO を用いてワールドトラッキングを実現しているのであれば，この場合でも CG 画像は回転するはずである．したがって，ARKit は VIO を用いてワールドトラッキングを実現していることがはっきりした．

図 3.6 端末ごと 90° 回転移動した場合

図 3.7 端末を固定し，実世界の平面を 90° 回転させた場合

3.2 画像認識

3.2.1 画像認識の理論

ARKit における画像認識は，対象となる画像（平面物体を正面から撮影した画像）を事前に登録することで可能になる．準備した画像とほぼ同じ大きさと方向で撮影された場合だけでなく，近づいたり離れたりすることによる被写体像の拡大縮小や，斜め方向から撮影したことにより像が歪んだ場合でも認識可能である．このようなことを可能にする画像認識の理論について考察した．

最も単純な画像認識技術はテンプレートマッチングである．これは，登録画像を入力画像内でラスタスキャンすることで，最も類似する部分を検出するアルゴリズムである．類似度として正規化相互相関法などを用いるので，許容される違いは全体的な明るさ変化と若干のノイズ程度である．大きさや回転角度が変化する被写体をテンプレートマッチングで認識させるには，そのように変形したテンプレートを準備する必要があるが，計算時間が膨大になる割に効果が少ない．

大きさと回転の変化に強い画像特徴を用いることで，これらに頑健な画像認識が可能になる．そのような画像特徴として SIFT 特徴[3]がある．SIFT 特徴は，コーナー特徴を拡張したような画像特徴である．しかし，SIFT 特徴は対象物を斜めから観察した場合に生じる射影歪みにはそれほど頑健ではない．大きさと回転に加えてアフィン変換に不変な画像特徴として Hessian-Affine，Harris-Affine，MSER などが提案されている[4]．ARKit はこのような変形に強い画像特徴を使って画像認識を実現している可能性が高い．

3.2.2 画像認識の実験

ARKit で画像認識を行う場合，認識したい画像を用意し，フォルダに入れておく必要がある．図 3.8 に用意した画像の一例とその認識結果を示す．

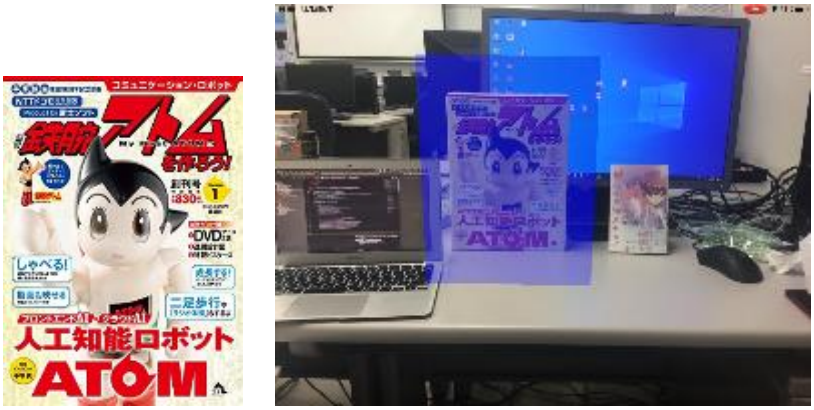


図 3.8 登録した画像（左）と，画像認識の結果（右）の例

画像認識のためのコードセグメントを図 3.9 に示す.

```
1    let configuration = ARImageTrackingConfiguration()
2    configuration.trackingImages = ARReferenceImage.referenceImages
                                    (inGroupNamed:”AR Resources”,bundle:nil)!
3    sceneView.session.run(configuration)
```

図 3.9 画像認識のためのコードセグメント

1 行目で ARImageTrackingConfiguration クラスのインスタンスを生成し，変数 configuration に代入している．ARImageTrackingConfiguration のメンバ変数で，画像認識で主に使用する変数は

・trackingImages

・maximumNumberOfTrackingImages

である．これらの変数に，認識したい画像と認識対象の数の上限を設定する．

2 行目でメンバ変数 trackingImages に検出したい画像のグループをセットしている．ARReferenceImage.referenceImages(inGroupNamed:”AR Resources”,bundle:nil) はデフォルトでアクセスするフォルダが決まっており，“Assets.xcassets” にアクセスする(これはプログラムの仕様上初めから指定されている)．その中の “AR Resources” グループを検出対象としている(図 3.10)．referenceImages はクラス ARReferenceImage のメソッドで，クラスメソッドである(クラスメソッドはインスタンス化しなくてもコールできる)．返り値は Set<ARReferenceImages>である。

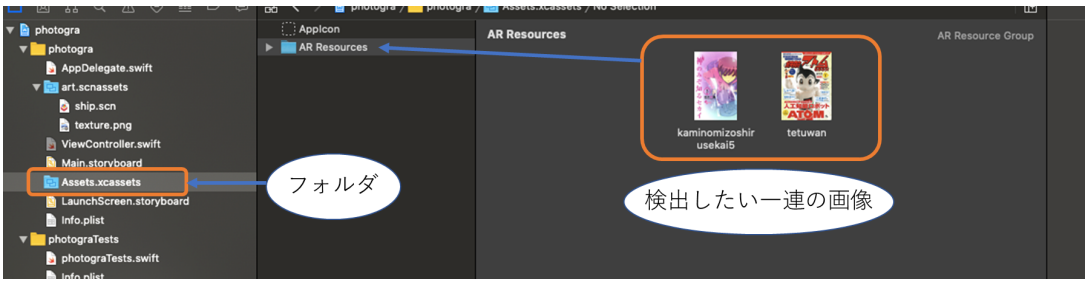


図 3.10 検出したい画像の保存場所

Swift は bundle を使用して，アプリ，フレームワーク，プラグイン，および他の多くの特定の種類のコンテンツを表すことができるが，画像認識や物体認識では bundle を必要としないので nil となっている．

検出対象はグループ内の全画像となっているため登録した画像をプログラムで指定する必要がない．特定画像のみを検出したい場合，プログラムで特定画像以外は検出できないようにするか特定画像以外をグループに含めないといった工夫が必要である．

３行目は平面検出と同様で，画像を認識するように設定した configuration を sceneView に適用して実行させている．

今回の実験で検出する画像を図 3.11 に示す．これらは，サイズが違う２枚のカラー画像である．

図 3.11 画像認識の実験に用いた２枚の画像

（左：画像小(480×642)、右：画像大(506×798)）

まず，画像認識できる被写体の大きさの範囲を求めた．あらかじめ行った予備実験で，画像大であれ，画像小であれ，被写体が適当な大きさに撮影されておれば認識できることを確認した．図 3.12 に，それぞれの被写体単独で認識できる最も大きな撮影状態と最も小さな撮影状態を示す．(a)は画像小を約 2m から撮影した．(c)は画像大を約 1m から撮影した．(b)と(d)はディスプレイの横幅いっぱいに，画像の横幅のやや内側が表示されている．これらから次のようなことがわかる．

① 最も小さく撮影されて認識できる限度は，ディスプレイ横幅の 10%程度である．しかし，その度合いは画像によって変わる．顕著な特徴が多い画像小は，撮影された大きさが小さくても認識できる傾向にあり，顕著な特徴が少ない画像大は，撮影される大きさが大きても認識に失敗する傾向にあったと考えられる．

② 被写体の一部しか写っていない場合でも，画像短辺の 90%程度が撮影されていれば認識可能である．

(a)画像小を 2m から認識

(b)画像小を横幅よりも内側で認識

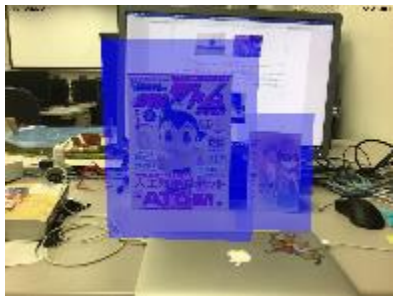
(c)画像大を 1m から認識

(d)画像大を横幅よりも内側で認識

図 3.12 画像認識できる最も小さい撮影状態と最も大きい撮影状態

次に,画像大と画像小を同じ距離から同時に認識させた.その結果,図 3.13(a)では,画像小だけが認識でき，図 3.13(b)では両方が認識できた．(a)は画像大が認識可能な画面上でのサイズをやや下回ったものと推察される．

(a)同時認識で一方の認識に失敗した場合



(b)同時認識で両方の認識に成功した場合

図 3.13 複数画像の同時認識

最後に被写体を斜めから撮影した場合や回転させた場合の認識性能を確認する実験を行なった．平面的な被写体を斜めから撮影すると，遠方が小さくなる射影歪み受けるため，画像認識が難しくなる．いくつかの結果を図 3.14 に示す．

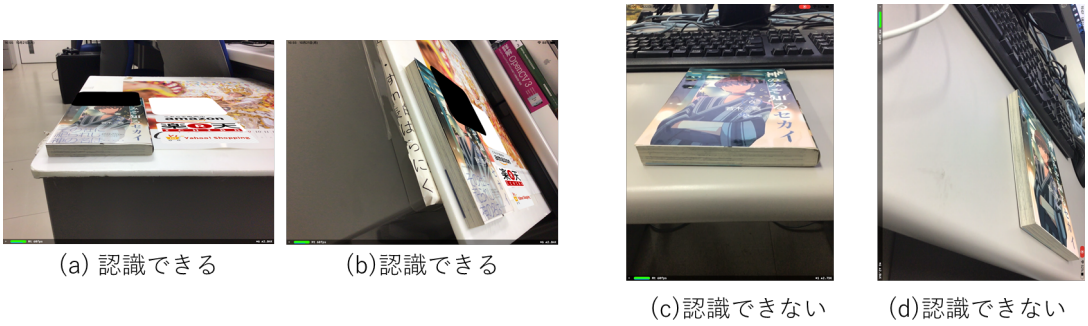


図 3.14 被写体を斜めや回転させて撮影した場合の認識結果

図 3.14(a), (b), (e)のようにかなりの角度をつけ，さらに回転させた状態でも認識が可能であった．(e)は書籍を正面から見たときを 0° として右に 90° ，左に 90° ，180° 回転させた状態である．これらの成功例には，認識した画像の上や横に，黒い長方形や広告の長方形バナーを表示させている．これらの長方形が認識した書籍の形状に合わせて変化していることから，被写体の姿勢が認識できていることがわかる．

(c), (d)は認識できなかった．(c)は変形の度合いが小さいので認識されてもおかしくないが，表面での光の反射が強く，それが影響していると考えられる．(d)は特徴点のマッチングが不可能なほどに変形が酷かったと考えられる．

3.3 物体認識

3.3.1 物体認識の理論

ARKit の物体認識は，事前に形状測定した立体物を，撮影している映像の中から認識する機能である．立体スキャナアプリが 3 次元形状を point cloud として保存していると考えると，物体認識のアルゴリズムは，概ね，次のようになる．

① スキャナアプリで立体物の形状を測定し，その形状を point cloud として登録する．
② シーンを撮影した画像から point cloud を測定し，登録した point cloud とマッチングさせる．マッチングに成功すれば，物体が認識できたと判断する．

3.3.2 物体認識の実験

物体認識させる立体物して図 3.15 のルービックキューブとシカのフィギュアを用いた．これらを立体スキャナアプリ“Scanning and Detecting 3D Objects”で測定して，実験に用いた．

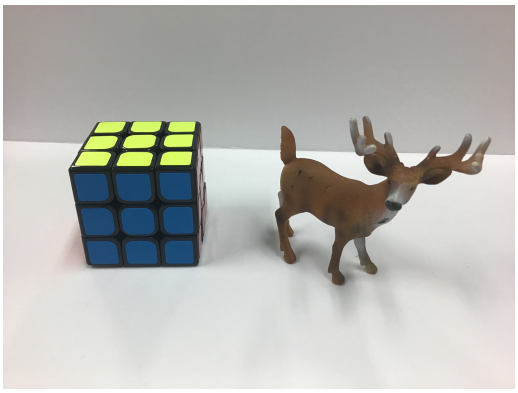
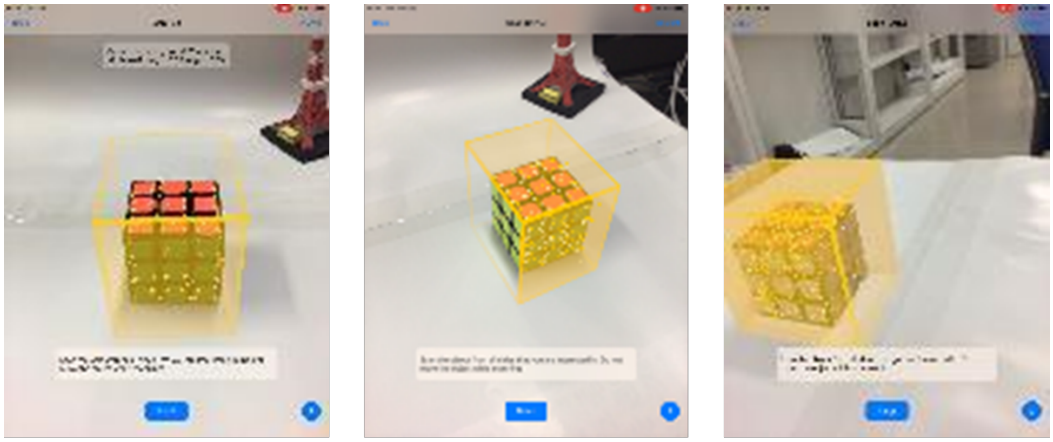


図 3.15 物体認識に用いた立体物

図 3.16 にスキャンの様子を示す．まず対象物を無色の平面の上に置く．すると，対象物体の周囲にバウンディングボックス（物体の縦横奥行をカバーする直方体）が表示される．今回は下図のようにルービックキューブを正面，側面，後面の方向から撮影した．その結果を“rubic.arobject”のような名称で，デフォルトで存在する“Assets.xcassets”フォルダに保存する．なお，無色の平面ではなく，画像特徴の多い平面の上に対象物を置いて測定すると，平面も対象物の一部として測定してしまうので，物体認識が正しく動作しない．



(a)前方からスキャン　(b)斜め方向からスキャン　(c)後方からスキャン

図 3.16 対象物体をスキャンする様子

次に，物体認識のためのコードセグメントを図 3.17 に示す．

```
1    let configuration = ARWorldTrackingConfiguration()
2    configuration.detectionObjects = ARReferenceObject.referenceObjects
                                        (inGroupNamed:”AR Resources”,bundle:nil)!
3    sceneView.session.run(configuration)
```

図 3.17 物体認識のためのコードセグメント

処理の流れは画像認識とほとんど変わらない．1 行目で平面検出と同じ ARWorldTrackingConfiguration のインスタンスを生成し，変数 configuration に代入している．2 行目では，メンバ変数 detectionObjects に検出したい物体のスキャンデータを画像認識と同じように代入する．スキャンデータは，先に説明した画像認識で使用したフォルダとグループに入れておけば物体認識可能になる．3 行目で，表示する sceneView に設定した configuration を適用して実行させている．

まず，物体認識によって立体物の位置と姿勢が認識できていることを確認する実験を行った．そのために，図 3.18 のように机の上に立体物（シカ）をおき，これを移動させたり，iOS デバイスを持って視点を移動させた場合の認識結果を調査した．

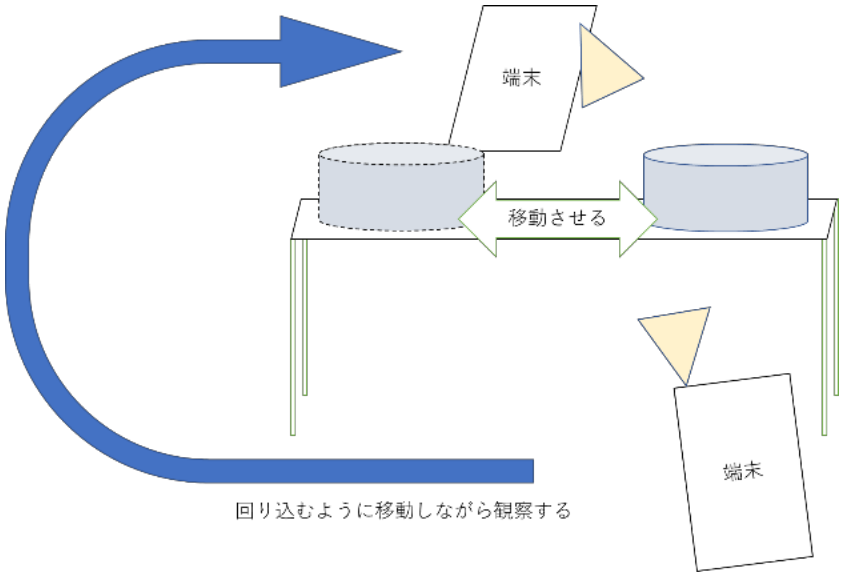
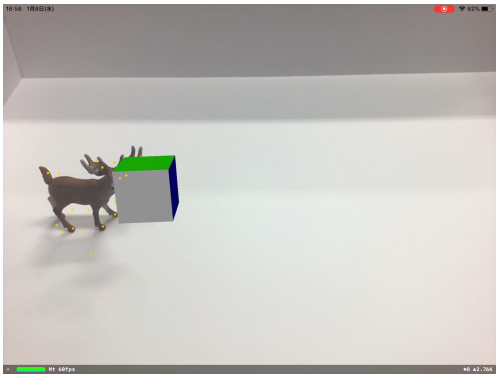
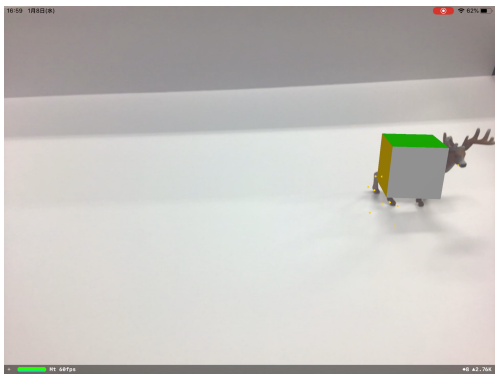
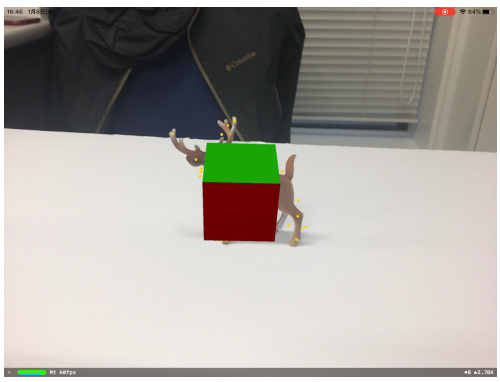
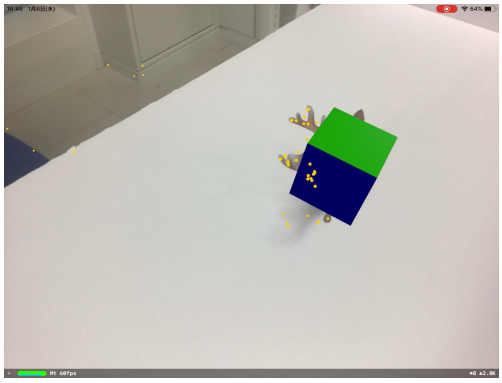


図 3.18 物体認識の実験方法

認識結果を確認するために，6 色で塗り分けた立方体を認識結果の位置・姿勢で表示させた．その結果を図 3.19 に示す．iOS デバイスを固定して立体物を平行移動させた場合，認識結果を示す立体物が概ね正しく平行移動した．次に，対象物体の周りを移動し，45°，90°，裏面から観察した．360°どこから見ても異なる形をしているシカのフィギュアを，どこから観察しても認識できた．

(a) 対象物を移動させた場合の認識結果

(b) iOSデバイスの視点を回り込むように変化させた場合の認識結果

図 3.19 物体認識の認識結果

次に ARKit が物体認識に色情報を利用しているかどうかを検証した．ルービックキューブのように対称性が高い物体は，形状だけで認識すると姿勢に曖昧性が生じる．色情報を利用していれば，そのような問題を解決できる．

図 3.20 左のように，ルービックキューブの色がそろった状態でスキャンし，登録した．その後，ルービックキューブをランダムにシャフルし物体認識させたが，認識しなくかった．このことから，物体認識に色情報を利用していると考えられる．


図 3.20 登録したルービックキューブ（左），認識させたルービックキューブ（右）

4 章 ARKit を使った AR のデモ

検証した ARKit の認識機能を利用して，２つのデモアプリを作成した．第一のデモアプリは「物体認識による移動物体のコース制御」，第二のデモアプリは「画像認識を利用した販売促進」である．

4.1 物体認識による移動物体のコース制御のデモアプリ

図 4.1 に，製作したデモの１シーンを示す．スタート地点とゴール地点に“Start”，“Goal”の文字列を貼り付け，それらを画像認識させる．デモアプリを開始させた時に，スタート地点に移動物体（車）の CG 画像を，ゴール地点に家屋の CG 画像を表示させる．スタートボタンを押すと，あらかじめ決められた逆 S 字のコースを車が移動し，ゴールに達する．移動軌跡も重畳表示した．

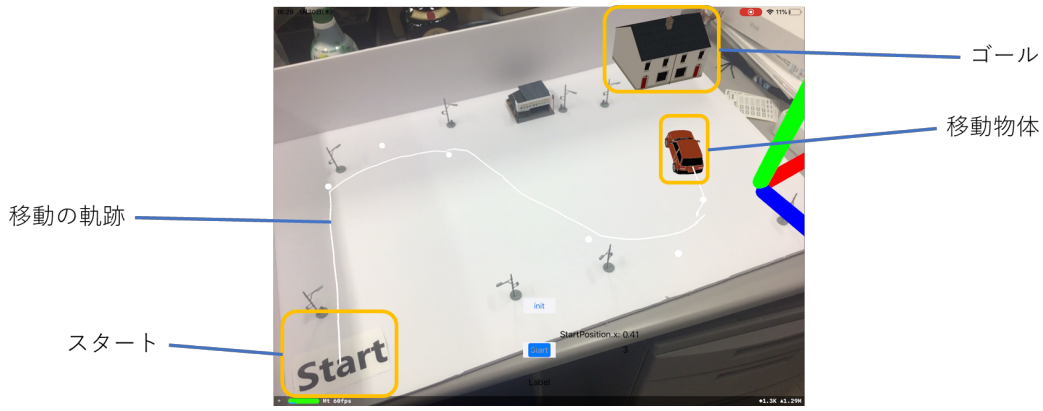
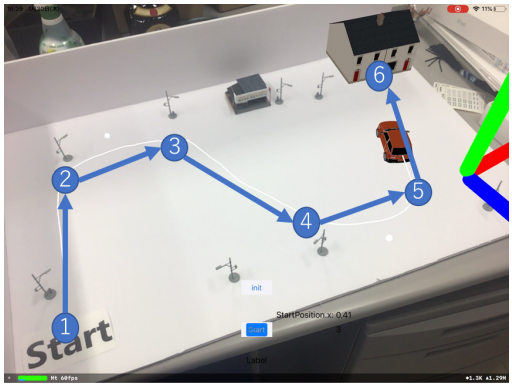


図 4.1 制作したデモの１シーン

車の基本的な移動動作は，図 4.2 のように，1 番から 6 番のマーカーを順に通過することである．車の移動経路が滑らかになるように，軌道の生成にベジエ曲線を利用した．



コースにマーカーを用意し、番号を振る。
その番号順に移動物体は移動する

図 4.2 移動物体の基本動作

コース上に障害物を検知すると 2 つのパターンで避けるようにプログラムした．一つはコース自体を変えた．図 4.3 左の場合であれば，本来は 1→2→3 のパターンで進むコースを 1→3 で進むコースに変えた．他の場合は，図 4.3 右のように，障害物を検知するとそれを回り込むように動くパターンである．

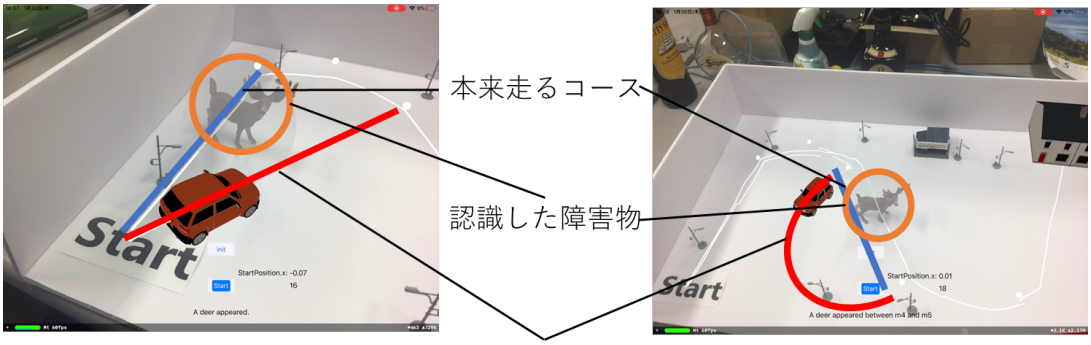


図 4.3 障害物検知と避ける 2 つのパターン

ARKit に障害物との衝突を検知する方法はないので，バウンディング球を使う方法を実装した．車の移動に関するアルゴリズム全体は，以下のようになる．

① 基本的に車は決められたコースを走る
② ARKit で障害物を検知するとバウンディング球を設定する
③ 予め設定した移動軌跡とバウンディング球が交差すると衝突と判定する
④ 移動物体は回避行動を行う
⑤ ①～④を繰り返し，最終的にゴールする

障害物の回避行動は図 4.3 が示すように二つのパターンが存在する．(a)のパターンは、図 4.2 において 1→2, 2→3 の間で障害物を検知した時に行う。それ以降の 3→4, 4→5, 5→6 で障害物を検知した場合(b)のパターンで回避を行う。

4.2 画像認識を利用した販売促進のデモアプリ

ARKit による画像認識は，書籍を持つ手や指が多少邪魔をしたり，書籍を斜めから撮影しても認識できるなど，実環境での使用に頑健である．この特徴を利用して，販売したい物体を認識すると EC サイトに誘導したり，広告表示などを行うデモアプリを作成した．

図 4.4 に，作成した販促アプリのデモ画面を示す．EC サイトの画像を画面でタップ

すると EC サイトに誘導される．動画像や広告，場合によっては 3D 物体などを配置することが可能である．このような発想は特別なものではないが，iOS 端末と ARKit の画像認識は販売促進との相性が良く，かなり使いやすいと感じる．

図 4.4 画像認識による販売促進アプリ

購入するものが電子書籍であれば，iOS 端末で書籍情報を画像認識し，電子書籍を EC サイトで購入し，即座にその端末で読むことができる（図 4.5）．書籍の情報を検索するという従来の方法よりシームレスに EC サイトに誘導できる．これはカメラがインターネットと繋がっていること，画像を処理できるフレームワーク，描画機能がある iOS 端末ならではのことである．

図 4.5 デモアプリでの電子書籍の購入と従来方法の比較

5章　結論

本論文では ARKit の認識機能について調査し，それらの機能を使ったデモアプリを作成した．その結果，ARKit が非常に優れたツールであることを理解できた．Visual-inertia odometry を使って，実写映像に CG 映像を違和感なく重畳することができる．画像認識機能は，書籍などの平面被写体を斜めから撮影したような場合でも認識可能である．物体認識機能を使うことで，立体物をあらゆる方向から認識することができる．

ARKit が実現している平面認識，画像認識，物体認識は高機能かつ高性能であり，これがなければ容易に実現できるものではない．また，AR の入門ツールとして最適である．例えば，水平面を認識するために，SfM で３次元点群データを得，それに最小二乗法や PCA を適用することは，かなりの経験と実験が必要である．しかし，ARKit を使ったソースコードでは，わずか 3 行程度で確実な平面検出の設定が可能である．

ARKit は，関連の最先端技術を集約させたものではあるが，内部処理はブラックボックスになっており，どういう処理をしているのかは明らかにされていない．平面認識や画像認識，物体認識などのアルゴリズムを十分に理解し，真に AR の技術を理解したいのであれば，他のプラットフォームや原理を調査しなければならない．

AR のデモンストレーションや AR プロジェクトを短時間で実装したい場合には，ARKit は最適な AR プラットフォームであり，AR の開発やプロモーションに大いに貢献する技術である．

参考文献

[1] バンダイチャンネル 「電脳コイル」のワンシーン, https://www.b-ch.com/titles/1097/001

[2] https://developer.apple.com/jp/documentation/arkit/understanding_world_tracking_in_arkit/

[3] Adrian Kaehler,Gary Bradski「詳解 OpenCV3 コンピュータビジョンライブラリを使った画像処理・認識」, オライリージャパン pp.520 - 525

[4] 藤吉 弘亘,「画像局所特徴量 SIFT とそれ以降のアプローチ」, MIRU2013 チュートリアル, https://www.slideshare.net/hironobufujiyoshi/miru2013sift

[5] Erin Pangilian, Steve Lukas and Vasanth Mohan, Editors, “Creating Augmented & Virtual Realities,” O’Reilly, 2019.

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

[collisionTest(viewController.swift)]

Xcode にて作成した「物体認識による移動物体のコース制御」デモアプリである．描画機能 SceneKit を使用して移動物体などを描画している．

[collisionTest(collision.swift)]

バウンディング球の設定や各種ベクトルなど，障害物の検知に必要な計算を記述しているプログラム

[ad_gain(viewController.swift)]

Xcode にて作成した「画像認識を利用した販売促進」デモアプリである．EC サイトのバナーを実際にタッチすると指定の EC サイトに誘導される．