



自己位置推定結果の活用-iPadのセンサ情報を併用した道案内システム-

M2 田川幸汰

1 前回までの結果

屋内シーンにおける特徴量の乏しい領域にポスターを貼付することでテクスチャ情報を補強し、その結果を用いて高精度な3次元モデルを生成した。このモデルと入力動画の各フレームを対応付けることで自己位置推定を実施し、さらに推定された位置・姿勢情報を基に、直進・右左折などの案内情報をCGとしてフレーム上に描画することで、AR道案内システムとしての基本的な処理フローを構築した。

また、アイコンが正しく表示されている様子から、特徴点マッチングが適切に行われている場合には、自己位置推定の精度が実用上問題ないレベルに達していることを確認した(図1)。

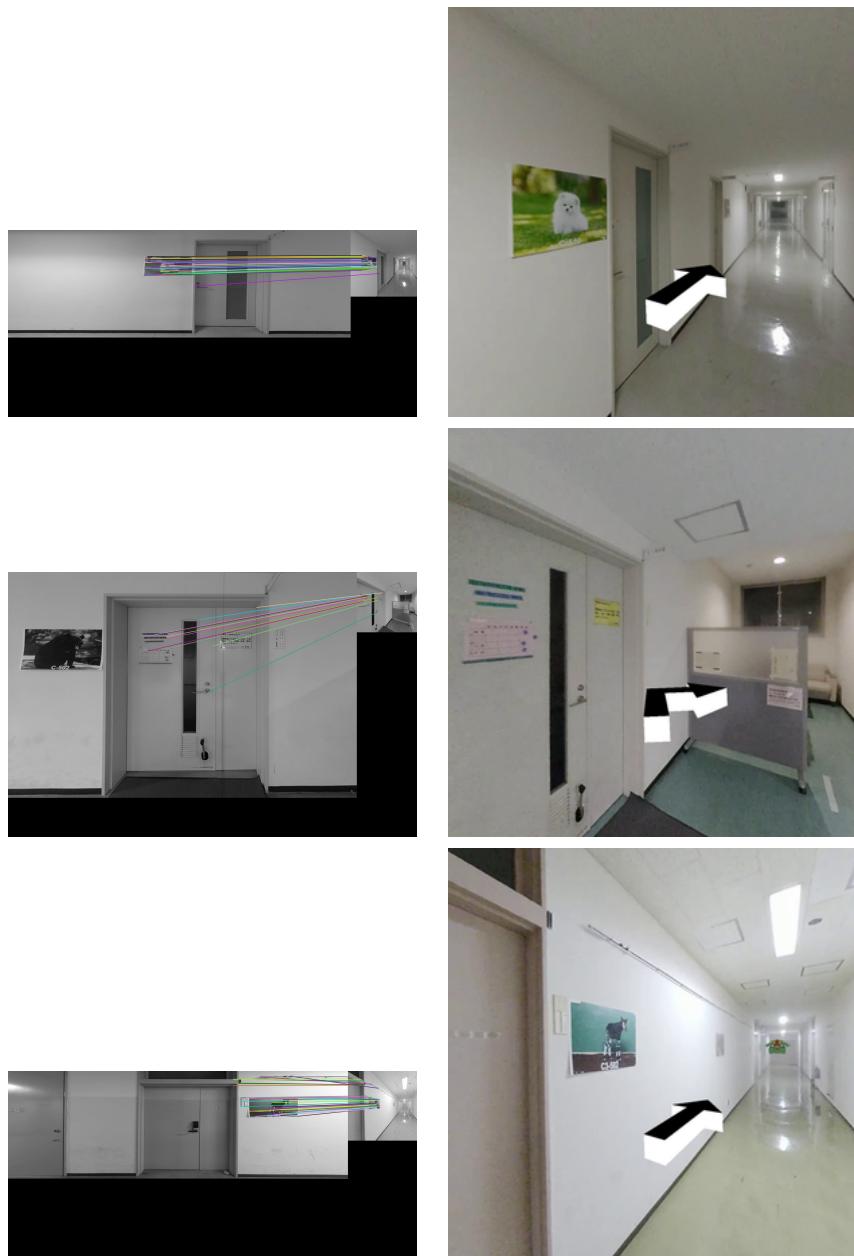


図1: 実行結果



2 概要

前回までは、動画データに対してCGを重畠する形でAR道案内システムの処理フローを構築した。しかし、実際のカメラ入力を用いてリアルタイム動作を行う場合、特徴点マッチング処理による遅延が発生し、自己位置推定結果が実際の端末位置とずれてしまう可能性がある。また、屋内環境では常に十分な特徴量が存在するとは限らず、マッチングに失敗して自己位置が適切に更新されない場合も想定される。

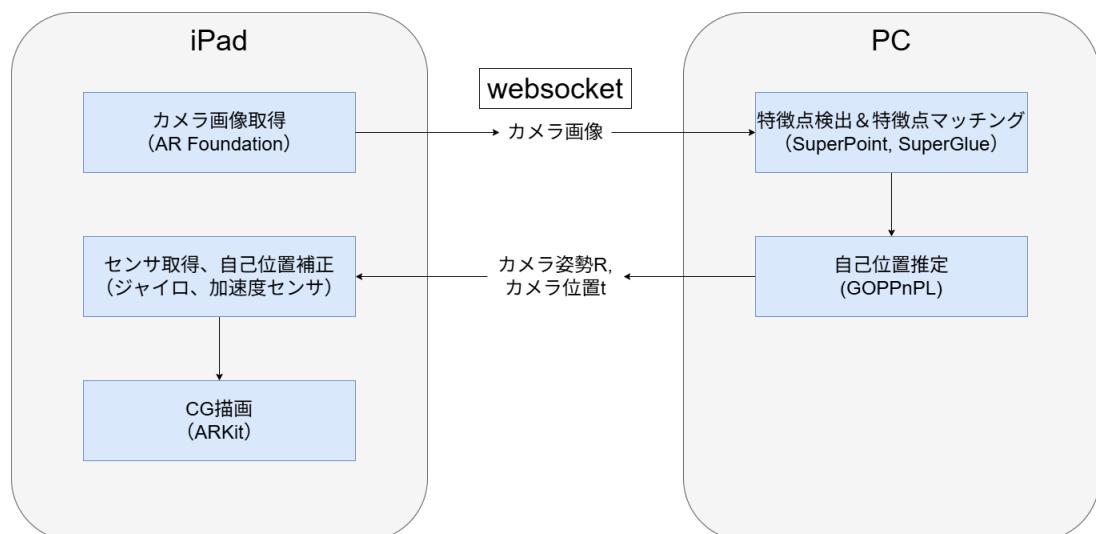
これらの課題を踏まえ、より安定した自己位置推定を実現し、実用的なAR道案内システムを構築するために、センサ情報の併用を検討することとした。本報告では、ユーザー端末としてiPad Pro(第5世代)を使用し、以下の構成でシステムを実装した。

1. iPadのカメラ映像をPCに送信
2. PC側で特徴点マッチングおよび自己位置推定を実行
3. 推定結果をiPadに送信
4. iPad上でCGアイコンを描画し案内情報を提示

本システムを用いて、推定位置に基づいたCGが想定した場所に正しく描画されるかを検証する。

3 AR道案内システムの概要

本研究で構築したAR道案内システムの全体構成と処理の流れについて説明する。システム概要図を図??に示す。本システムは、ユーザー端末(iPad)とPCの二つのモジュールで構成されており、両者はWebSocketによる双方向通信によりリアルタイムでデータをやり取りしている。WebSocketは、一度接続を確立するとクライアント・サーバー双方から自由にデータを送受信できるため、低遅延での通信が可能となる。



3.1 モジュール構成と主な処理

- iPad

- 目的地の設定
- カメラ映像の取得
- 端末センサ情報（加速度計、ジャイロ）の取得
- 自己位置とセンサ情報を融合して端末の位置・姿勢を計算



- CGアイコンの描画
- PCにカメラ映像と自己位置情報を送信

- PC

- iPadから送信されたカメラ映像と自己位置情報の受信
- 受信情報に基づき、近傍テクスチャの検索
- SuperPoint, SuperGlueによる特徴点検出およびマッチング
- 2D-3D対応付けによる自己位置の更新
- iPadに推定結果を送信

この構成により、計算負荷の高い画像処理や自己位置推定はPC側で行われ、iPad上ではセンサ情報とPCからの推定結果を組み合わせてリアルタイムに案内情報を表示できる。

4 iPad側の処理

ユーザー端末として使用するiPad上で行われる処理について説明する。

4.1 アプリ起動時

アプリ起動時にAR環境とユーザー操作環境を初期化する。

- **SensorARApp.swift** アプリのエントリーポイント。起動時に最初に呼ばれ、表示する画面としてContentViewを設定する。
- **ARViewControllerRepresentable.swift** UIKitのARViewControllerをSwiftUI上に埋め込むためのラッパー。SwiftUIからAR画面を扱えるようにする。
- **ContentView.swift** アプリメイン画面。AR画面の表示と、Start AR / Stop ARボタンによる操作を提供する。

4.2 端末の自己位置の管理とAR表示

ARViewController.swift はAR表示と自己位置推定を管理する。主要な処理は以下の通りである。

- **start()** Start ARボタンを押すと実行される。目的地をマップ上から選択する処理を呼び出し、選択が完了したらARセッションを開始する。
- **main()** ARシーンの初期ノード(球体、矢印)を作成し、ARセッションを開始する。AR表示は、ARKitとSceneKitの2つのライブラリを統合したARSCNViewで管理され、カメラ映像上に3Dオブジェクトを重ねて表示する。さらに、PCとのネットワーク接続と初回のカメラフレーム送信も実行する。
- **stop()** Stop ARボタンを押すと実行される。ARセッションやノード描画、マップUIなどの各種処理を停止する。
- **updateModelPose()** 画面更新時(renderer実行時)に呼び出される。PCから受信したカメラ姿勢と端末のIMU情報を組み合わせて端末の位置姿勢を計算し、オブジェクトの位置と姿勢を更新する。球体ノードは目的地に描画され、矢印ノードは目的地の方向を向くようにカメラ前方1.0mに描画される。
- **captureCameraFrame()** ARKitのカメラフレームを取得して画像に変換し、ネットワーク経由でPCに送信する。カメラの内部パラメータ、解像度、送信時刻も取得する。
- **updateCameraPoseFromPC(R:t:)** PCから受信した自己位置を適用する。初回受信時は自己位置を設定し、IMUによる自己位置推定を開始する。それ以降は前回送信時からのIMU積分値を組み合わせて自己位置を更新し、積分値をリセットする。なお、自己位置推定に失敗した場合は、IMUの積分値をリセットしない。適用完了後、カメラフレーム取得を呼び出す。



4.3 ルート選択

MapManager.swift マップ表示と目的地ルート選択を管理する。マップ画像を画面に表示し、ユーザーがタップで目的地や目的地までの経路上の右左折ポイントを選択できる状態にする。「選択終了」ボタンで選択を確定し、「戻る」ボタンで最後に追加したポイントを削除できる。選択したポイントはホモグラフィー行列を用いて画像座標から世界座標に変換する。

4.4 端末IMUによる自己位置補正

MotionManager.swift は端末の加速度計とジャイロセンサを用いて自己位置推定の補正を行う。端末の IMU 情報を定期的に取得し、角速度から回転行列を、加速度から位置変化を計算して前回値に加算して更新する。IMU 情報の取得には `CoreMotion` ライブライアリを用いる。

4.5 PCとの通信管理

NetworkManager.swift は iPad 側で PC との通信を管理する。WebSocket を用いて PC と接続し、カメラフレームや内部パラメータ、IMU 情報によって更新された自己位置を送信する。また、PC から送られてくる自己位置情報（回転行列と平行移動ベクトル）を非同期で受信し、AR 表示や自己位置推定に反映する。送信される画像は圧縮され、データは JSON 形式で構成される。

5 結果

本研究では、結果確認のために iPad から送信されるカメラフレーム画像を、事前に準備した別の画像に置き換えて処理を行った。図 2 に、処理で用いた入力画像を示す。

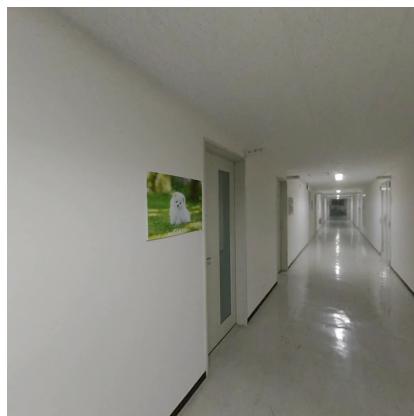
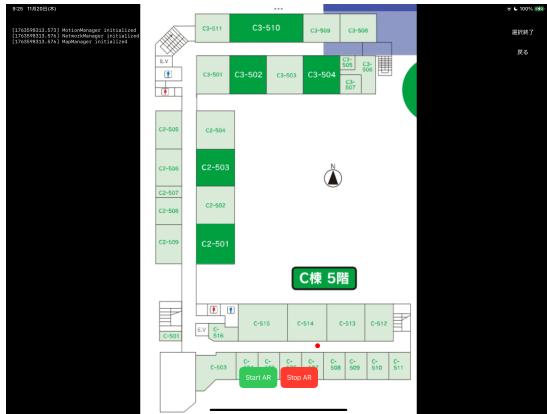


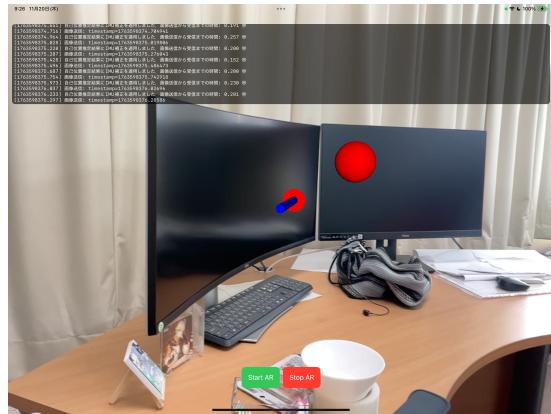
図 2: 結果確認のために使用した入力画像 (C-510 付近で撮影)

5.1 目的地に対するオブジェクト描画結果

2箇所の目的地を指定し、それぞれに対応する AR オブジェクトの描画結果を確認した。それぞれの目的地に対する AR オブジェクトの描画結果を、図 3、図 4 に示す。左側に入力画像、右側に対応するオブジェクト描画結果を並べて示すことで、目的地に応じてオブジェクトが正しく表示されていることを確認できる。

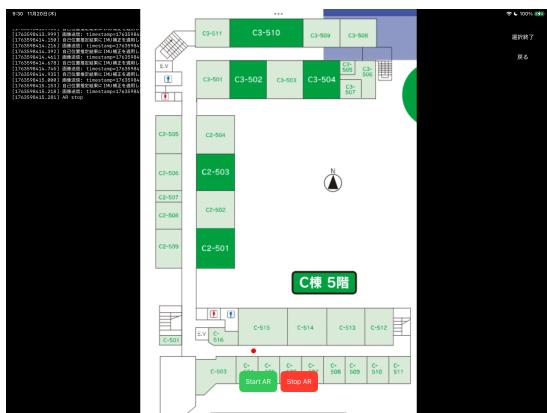


入力画像

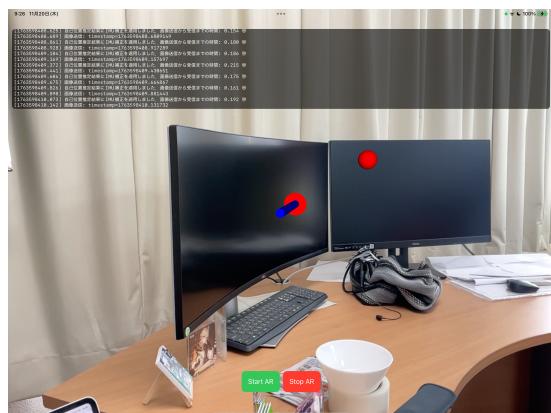


目的地1に対する描画結果

図3: 目的地1に対するオブジェクト描画結果



入力画像



目的地2に対する描画結果

図4: 目的地2に対するオブジェクト描画結果

より遠い場所を目的地として設定した方が、オブジェクトは小さく描画されていることがわかり、指定した任意の目的地に対して、対応するオブジェクトはある程度正しく位置に描画されていることを確認した。また、実行時の動画を([result_1120.mp4](#))に示す。

参考文献