



テクスチャ補完した3次元モデルの生成と自己位置推定

M2 田川幸汰

1 前回までの結果

テクスチャの特徴不足により特徴点マッチングおよび自己位置推定が困難となっていたため、特徴の乏しい面にポスターを貼付することでテクスチャ情報を補強した。この手法により、より豊富なテクスチャ情報を有する3次元モデルを生成し、その3次元モデルを用いて特徴点マッチングを実施した。マッチング結果を図1に示す。

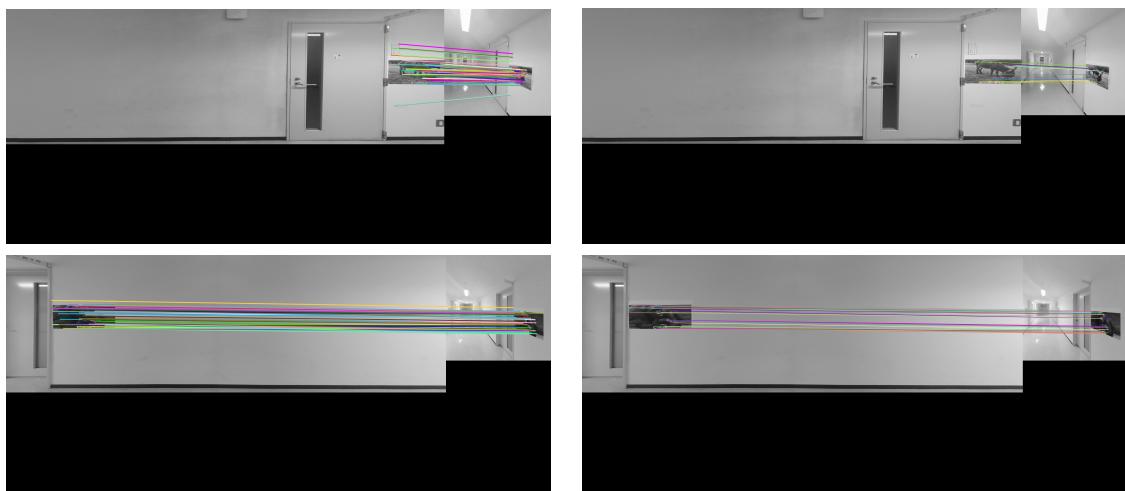


図1: 入力フレームに対する特徴点マッチング結果。左列は機械学習ベース手法、右列は従来手法

特徴点マッチングにSuperPointを用いた手法（左）では、テクスチャ補完のために貼付したポスターが写り込んでいるフレームにおいて、ほぼ全てのケースで正しくマッチングを行うことができた。一方、SIFTを用いた手法（右）では、カメラ近傍にポスターが写り込んでいる場合にのみマッチングが成立し、それ以外のフレームではマッチ数の不足や誤対応が生じる結果となった。さらに、一定時間以上連続して自己位置推定に失敗した際には、全データベース画像との再マッチングを行う仕様としたため、処理全体の計算時間に大きな差が生じる要因となった。

また、入出力を動画に対応させ、自己位置推定結果を活用する手法として2種類の出力を作成した。1つは、自己位置推定結果をフロアマップ上に表示するものであり(result_map.mp4)もう1つは、入力画像上に直進・右左折などの2Dアイコンを重畠表示するものである(result_2D.mp4)。

2 概要

前回の報告までに、特徴の乏しい面にポスターを貼付することでテクスチャ情報を補強した3次元モデルを生成したが、今回はその対象範囲をC棟5階全体へと拡張した。それに伴い、一部特徴点の再計測や全方位カメラの再撮影、ならびにカメラ位置推定のための2D-3D対応付けを実施した。

また、前回の報告では、自己位置推定結果を活用する手法として、入力画像上に直進・右左折などの2Dアイコンを重畠表示する、道案内システムのプロトタイプを構築した。今回は、アイコンや目的地などの要素をCGとして描画し、AR道案内システムとしての基本的な処理の流れを完成させた。



3 テクスチャ補完範囲の拡張

テクスチャの補完範囲をC棟5階全体に拡張した。3次元モデルを生成するまでの手順を以下に示す。

1. 特徴点の世界座標を計測

C棟5階南棟を原点として特徴点を計測する。計測にはレーザー距離計付き巻尺 GM5 (GOODMAN) を用いた。併せて、全方位カメラの撮影位置も計測する。撮影はおおむね4m間隔で行った。

2. テクスチャ補完に用いるポスターを貼付

歪みのない撮影を行うため、全方位カメラの正面にポスターを貼付する。ポスターはA2サイズであり、画像はWikimedia Commons [1]より取得した。

3. 全方位カメラによる撮影

カメラ位置（赤）およびポスター位置（青）を図??に示す。ポスター枚数の都合上、進行方向右側にのみテクスチャを補完した。また、カメラの高さは1.4mである。

4. 3次元モデルファイル (mqoファイル) の生成

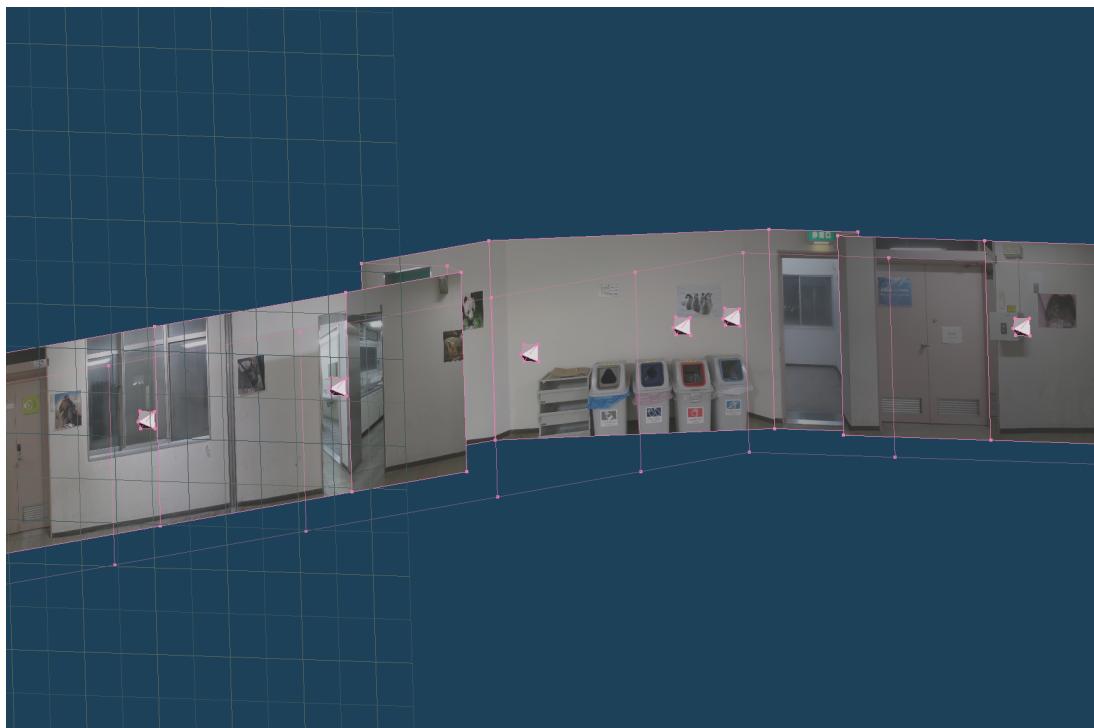
3次元モデルの頂点の世界座標、面の4隅の頂点番号およびUV座標を入力する。UV座標はテクスチャ割り当て時に修正するため、ここではテクスチャ4隅の頂点の世界座標から縦横比を求めて入力する。この一連の処理は、頂点の世界座標を入力することでプログラム内で自動的に実行される。

5. テクスチャの割り当て

各全方位カメラから複数視点の透視投影画像を生成し、それぞれの画像に対して2D-3D対応付けを行う。得られた対応付けとともにカメラ位置・姿勢の推定を行い、テクスチャ画像および切り出す四隅のUV座標を求める。



これら一連の処理により生成された3次元モデルの一部を図??に示す。若干のずれは見られるものの、概ね問題なくテクスチャが割り当てられていることが確認できる。



4 AR道案内システム

テクスチャを補完した3次元モデルを用いて自己位置推定を行い、その結果を活用した道案内システムを構築した。自己位置推定には岡本の手法(GOPPNPL)を利用しておおり、現状では点特徴のみを対応付けに用いている。ただし、同一平面上から3次元特徴を取得しているため、上下や左右が反転した自己位置推定結果が得られる場合がある。そこで、回転行列の符号($R[1, 2] < 0$)を参照し、カメラY軸が世界Z軸の負方向を示す、正しい天地方向の結果のみを採用している。

自己位置推定結果をもとに、現在位置から目的地までの道案内を行うシステムの一連の流れを以下に示す。

1. 目的地（ルート）を設定

フロアマップ上からルートを設定する。ルート検索機能は現時点では実装していないため、右左折のポイントも順番に指定する。

2. ウィンドウを作成

入力画像の縦横の長さ、画像の焦点距離を元に、入力画像を背景とするウィンドウを作成する。今回、CG処理にはOpenGLライブラリを用いている。また、ウィンドウやイベント管理はGLFWを用いている。

3. 3次元モデル読み込み

直進や右左折を示す矢印、目的地に描画する3次元モデルを読み込む（図2参照）。また、到着時やルート逸脱時に表示する文字テクスチャも生成する。

4. カメラおよびモデル位置姿勢の計算

自己位置推定結果に基づき、カメラおよび各モデルの位置・姿勢を計算する。各矢印は視点から1.0m奥、高さ0.5mの位置に描画される。直進矢印は次の目的地（右左折ポイント）方向を向くよう回転させ、目的地モデルは現在位置を向くように回転させる。

5. 3次元モデルを描画

現在位置とルート情報に基づき描画対象の3次元モデルを決定する。右左折ポイントから2.0m以内であれば右左折矢印を描画し、目的地から3.0m以内であれば画面下部に「目的地周辺です」とメッセージを表示する。それ以外の場合は直進矢印を描画する。

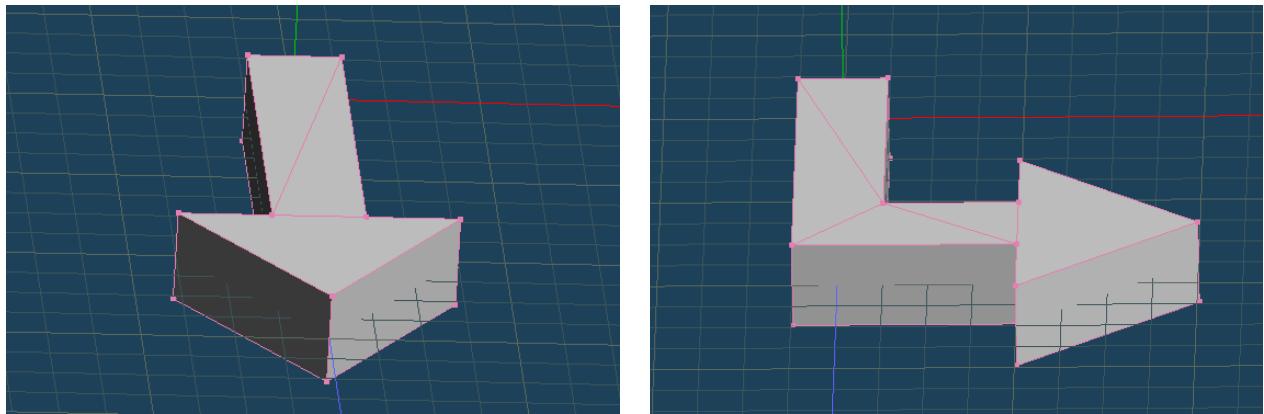


図2: 矢印の3次元モデル

4.1 実行結果

実行結果を図3に示す。また、動画出力は(result.mp4)に示す。右図は特徴点マッチングの結果、左図は当該フレームに対応するCG描画の結果である。特徴点マッチングが正しく行われた画像においては、CG描画も正しく反映されていることが確認できる。

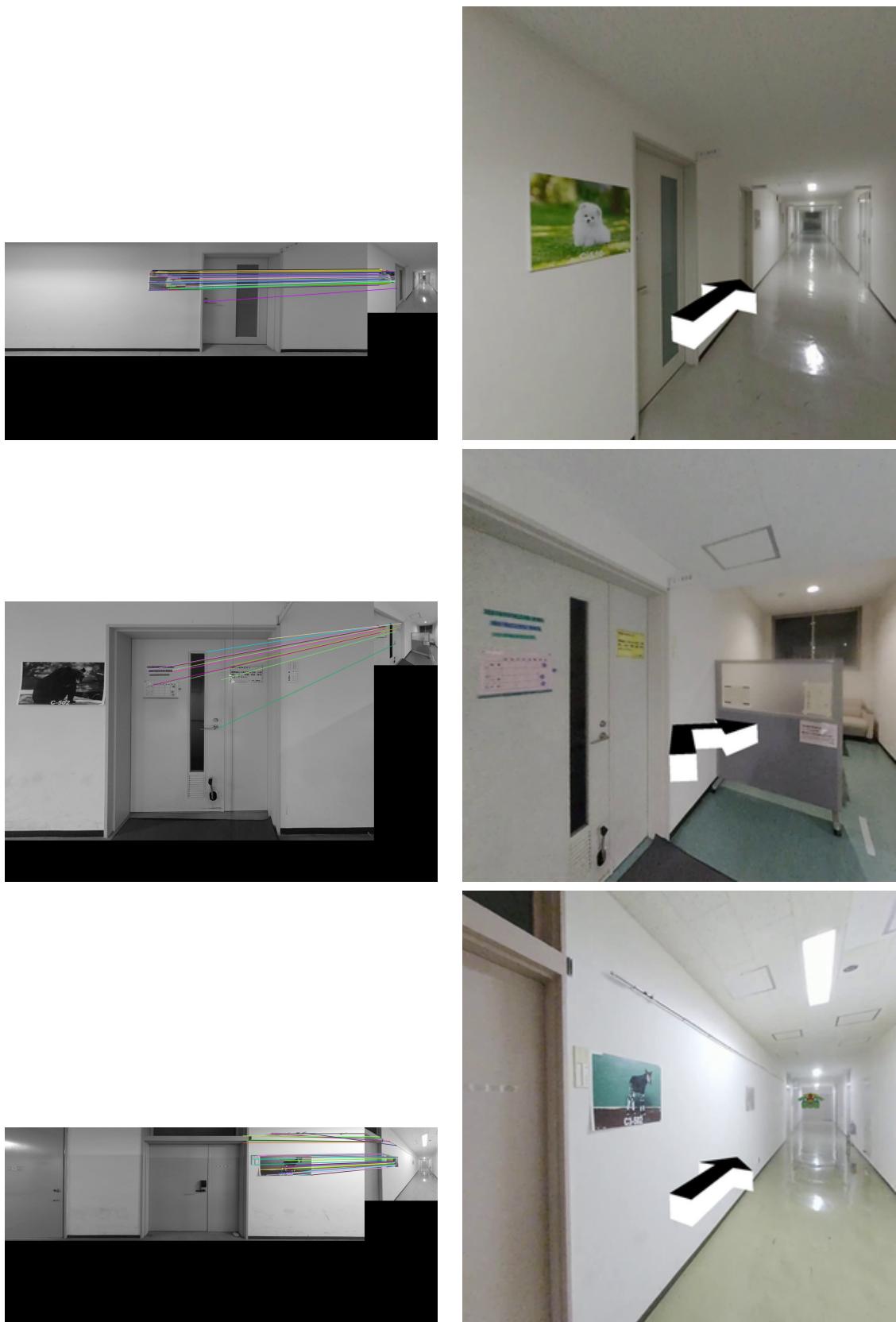


図3: 実行結果

また、特徴点マッチングに失敗したフレームとそのCG描画結果を図4に示す。局所的に類似する特徴を持つ別の画像と誤ってマッチングしたため、CG描画も正しく行われていない。このような誤マッチングを極力抑制し、仮に誤マッチングが発生した場合でも自己位置推定が誤っていると判断して位置を更新しない仕組みを導入することが望ましい。

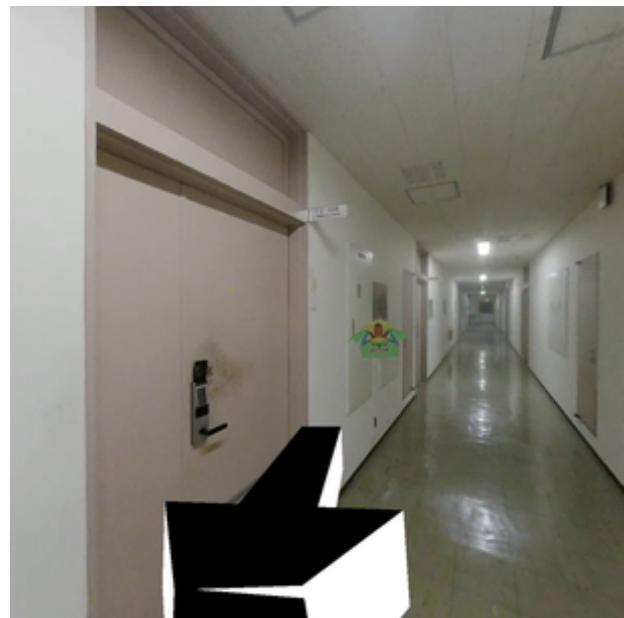
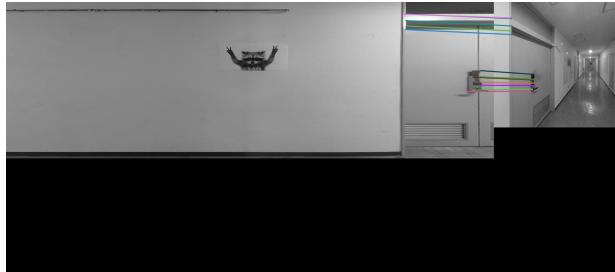


図4: 実行結果失敗例

実行時間については、現在位置がある程度把握されている場合、特徴点検出に平均0.015秒、特徴点マッチングに平均0.2秒、自己位置推定に0.01秒、CG描画に0.05秒を要した。したがって、1フレームあたりの処理時間は合計で約0.3秒となり、道案内システムとして利用する上で十分な実行性能であると言える。

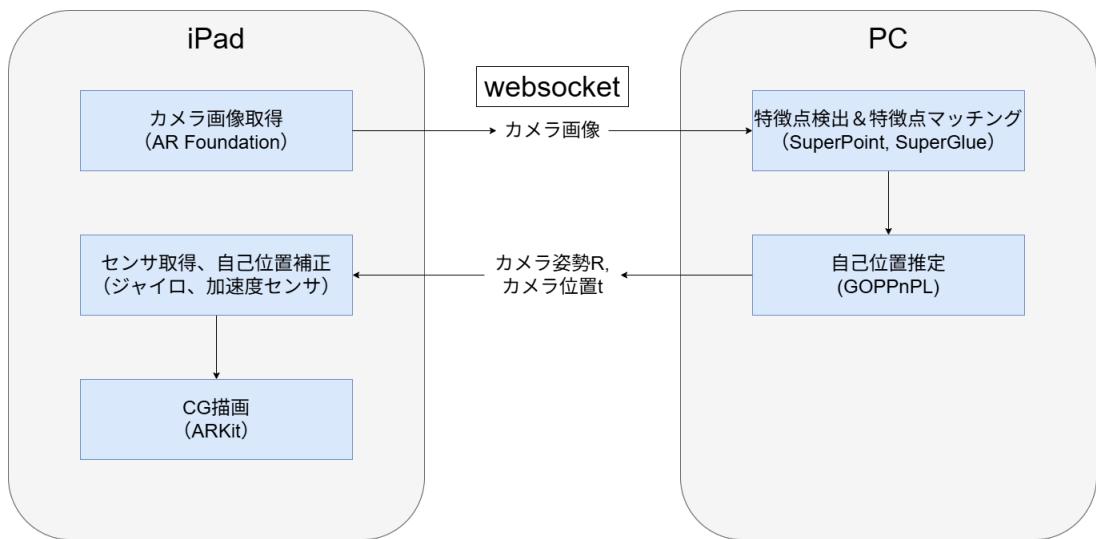
ただし、現在位置が不明な場合は、データベース内の全画像とマッチングを行う必要があるため、特徴点マッチングには約2秒を要する。このため、自己位置推定が不能な間も他の情報を用いて適宜自己位置を更新できれば、より安定した自己位置推定に基づく道案内システムを構築することが可能である。

5 今後の計画

今後の研究計画を以下に示す。

1. 11月以降：より安定した自己位置推定結果の取得と、AR道案内システムの改良

利用者端末のセンサ情報の活用：現状では、利用者端末のセンサ情報（加速度など）を用いて、前回の自己位置推定結果からの変位を算出する方法が最も実現しやすいと考えられる。このセンサ情報を基に、特徴点マッチングや自己位置推定の結果が正しいかどうかを検証したり、テクスチャ情報が乏しくマッチングできない場合の自己位置推定結果の補完として利用することで、自己位置推定の安定化につなげられると考えている。想定しているシステムの概要を図??に示す。現時点では、iPadを用いたカメラ画像の読み込み処理、センサ情報の取得処理、およびCG描画の基本的な実装を行っている ([iPad.mp4](#))。



参考文献

- [1] Wikimedia Commons contributors, "Wikimedia Commons, the free media repository." <https://commons.wikimedia.org/>. Accessed: 2025-08-27.