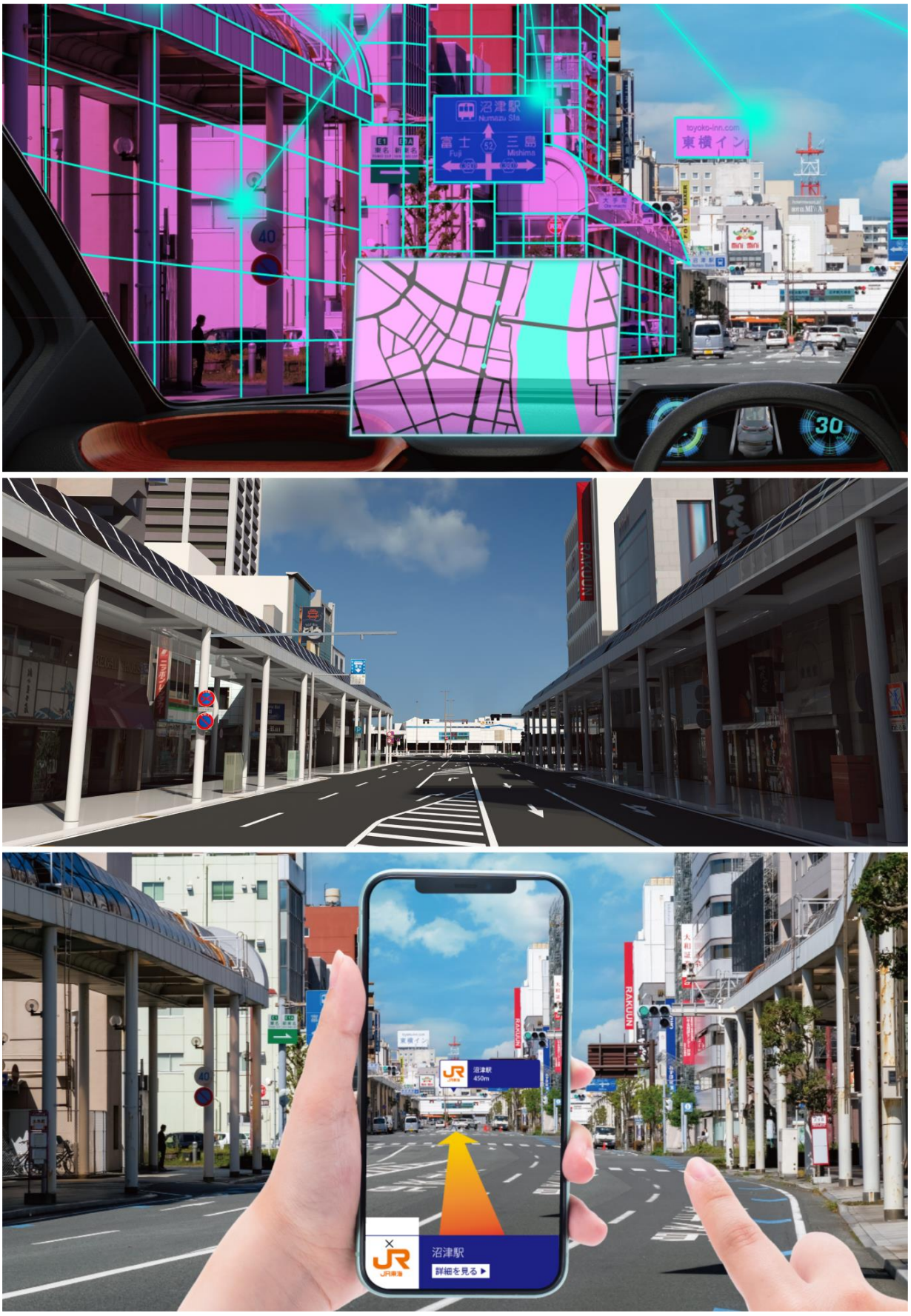
PLATEAU by MLIT

PLATEAU Technical Report

3D 都市モデル活用のための技術資料



3D 都市モデルに最適化した VPS の開発 v3.0 技術検証レポート

series No. 88

Technical Report on the Development of VPS Optimized for 3D City Models v3.0

目次

1. ユースケースの概要

1-1. 現状と課題

画像と 3D 地図等をマッチングさせることで位置を特定する VPS（Visual Positioning System）は、一般的には事前にユーザーが現地へ訪問し、撮影やスキャンによってマップを作成する仕組みとなっている。この VPS 用マップをオープンデータである 3D 都市モデルを活用して作成することができれば、汎用的かつスケーラブルな VPS を実現することができ、モビリティサービスやエンタメなど様々な分野へ応用が可能となる。

上記の実現に向け、2022 年度の開発「自動運転車両の自己位置推定における VPS 活用 Ver2」では、安価、簡易、安全、スケーラブルな自動運転システムの実現を目指し、産業技術総合研究所から提供された「C*」をベースに 3D 都市モデルと光学カメラ画像を組み合わせた VPS 技術の開発を行った。その結果、一定の条件下における自己位置推定には成功したものの、低速走行時の自己位置推定精度の低下やレンダリング処理の遅延等、サービス実装に向けた多くの課題が判明した。特に初期位置推定及び自律走行時における継続的なトラッキングに関する改善の必要性が明らかとなった。

1-2. 課題解決のアプローチ

今回の実証実験では、3D 都市モデルを活用した VPS 技術の確立と、社会実装に向けた汎用性の拡大の 2 つの観点で開発を行う。

1 つは昨年度の追加改修として、車両向け自律走行システムの高度化を目指す。具体的には、昨年度開発した「C*」をベースとする VPS 技術の処理能力向上を図るため、照合用のカメラ画像の分析に新たなセンシングシステムを統合するなどのトラッキング機能の安定性向上のための施策を講じるとともに、Kudan 社から提供を受けた VPS 技術である「KdVisual」を活用し、両 VPS を最適に組み合わせることで自己位置推定の精度とトラッキング機能の安定性の向上したシステムを構築する。
構築したシステムは自動運転車に搭載し車両の自動運転への利用可否を検証する。得られた自己位置推定位置の精度などは、高精度 GNSS の座標や自動運転システムの推定した座標と比較し評価する。
本システムが実現することにより、LOD3 の 3D 都市モデルが整備されている地域であれば、3D 都市モデルのみを入力とすることで、画像からの自己位置推定を行う VPS が利用可能となる。これにより、現実とデジタルデータの場所による照合が容易となり、都市域での安価な自己位置推定手段として AR の利用や電波条件の悪い場所での GPS の代替、都市内モビリティの自動運転の普及など、デジタルツインのインフラを支える技術となる。

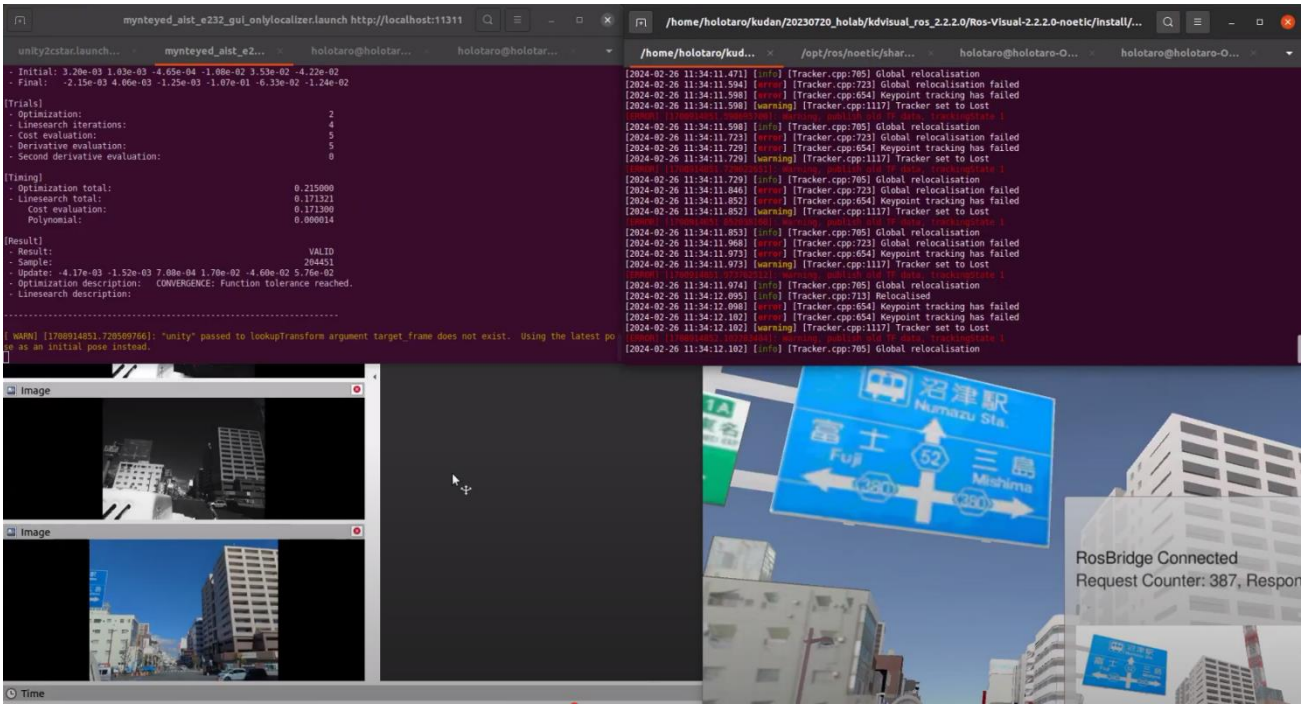


図 1-1 C*・KdVisual 動作画面

２つ目はスマートフォン向けのアプリケーションとして利用可能な VPS 技術の開発を目指し、プレティア・テクノロジーズ社が開発する「PretiaVPS」をベースに、3D 都市モデルを活用した自己位置推定システムを開発する。具体的には、3D 都市モデルから生成した点群マップと、スマートフォンで撮影したカメラ画像から

生成した点群マップとを比較し、自己位置を推定するシステムを構築する。さらに、利用者の移動量に応じた自己位置のトラッキング機能を構築することで、コンシューマ向けコンテンツの提供に利用可能な速度及び精度を担保する。

具体的には、AR ナビゲーションやラストワンマイル配送など、VPS を活用したサービス提供を検討する事業者が、事前のスマートフォンのスキャナアプリ等による空間スキャン不要で VPS を活用したシステムを開発できる環境を実現できる。従来の VPS システムでは、現実世界においてシステムを利用するすべてのエリアでスキャナアプリによる空間スキャンが必要であった。また、スキャンした空間マップをもとにシステムを開発するにあたり、マップに不備があった場合、開発者が何度もマップを作成し直す必要があった。これが VPS を活用したシステムやサービスの開発コストに大きく影響することも普及に向けた大きな課題となっている。本実証で開発するシステムを実現することで、これらの課題を開発し VPS 技術の社会実装を加速し、私たちの身近な生活の利便性に大きく貢献することができる。

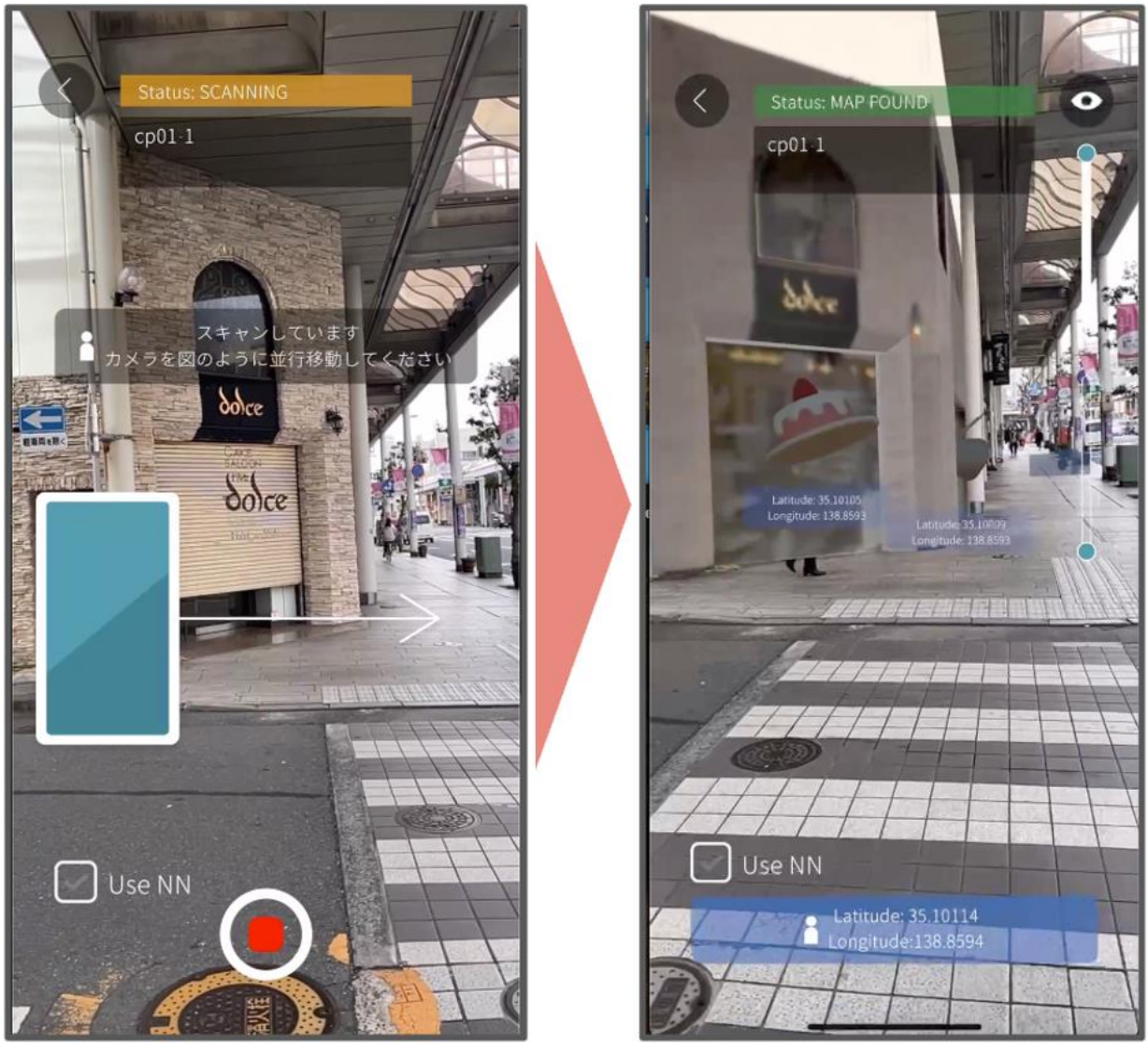


図 1-2 実現するスマートフォン向けアプリケーションのイメージ（AR ナビゲーション）

1-3. 創出価値

自動運転システム等に用いられている自己位置推定技術には、GNSS、LiDAR、速度計、ジャイロスコープ等の各種センサや、3D ベクトルデータ、3D 点群データ等の様々なデータが活用されており、一定の実用水準にあるものの、使用される LiDAR 等の機器は高額なものが多く、また、必要となる高精度なデジタルマップの作成負荷も高いため、様々な用途に導入しやすい状況には至っていない。また、GPS はビルが立ち並ぶ都市部や、屋内では精度が低下するといった問題もある。そのため、3D 地図と画像を照合することで環境に左右されず自己位置を特定することができる VPS 技術が着目されている。

今回の実証実験では、3D 都市モデルを基礎とするマップを用い、スマートフォンに搭載されているような汎用化されたカメラやプロセッサによる VPS を開発する。オープンデータである 3D 都市モデルのプリミティブな形状を VPS マップとして活用した自己位置推定システムが実装されることで、3D 都市モデルが整備されている地域であればどこでも簡易に自己位置推定システムの活用が可能となり、自動走行運転に限らず、観光や防災、まちづくりなどの様々な都市サービス導入が簡便になることで、デジタルツイン基盤としての活用につながることを目指す。

1-4. 想定事業機会

表 1-1 想定事業機会

項目	内容
利用者	● バス事業者、マイクロモビリティ事業者、自動運転車いす事業者 ● デベロッパー等まちづくり事業者 ● AR コンテンツ開発者
サービス仮説	● 例）小型モビリティに自動運転を付加することにより、利用者の満足度を高める。 ● 例）車載 AR サービスにつなげる。 ● 例）自己位置推定による座標取得により、各車両の位置の管理や運用管理ができる。
提供価値	● 各事業者のモビリティへの自動運転付加 ● モビリティサービスへのサービスインフラ提供 ● 都市における AR サービス開発環境の提供、VPS マップ作成コスト減による AR アプリ開発コストの削減

1-5. 本資料の構成

以下の項目に沿って、3D 都市モデルを活用した VPS 開発に関する車両向けシステム及びスマートフォン向けアプリケーション向けの有用性検証を行う。

表 1-2 本資料の構成

項目	目次	記載章
車両向けシステム	実証実験の概要	2
	実証システム	3
	実証技術の検証	4
	BtoB ビジネスでの有用性検証	5
スマートフォン向けアプリケーション	実証実験の概要	6
	実証システム	7
	実証技術の検証	8
	BtoB ビジネスでの有用性検証	9
両実証を踏まえた示唆	両実証の比較	10
	成果と課題	11

2. 車両向け自律走行システム：実証実験の概要

2-1. 実証仮説

- 今回の実証実験では、昨年度開発した「C*」をベースとする 3D 都市モデルと光学カメラ画像を組み合わせた VPS 技術の追加改修し、車両向け自律走行システムの高度化を目指す。具体的には、昨年度開発した「C*」の処理能力向上を図るため、照合用のカメラ画像の分析に新たなセンシングシステムを統合してトラッキング機能の安定性向上のための施策を講じるとともに、Kudan 社から提供を受けた VPS 技術である「KdVisual」を活用し、双方の VPS アルゴリズムを組み合わせることで自己位置推定の精度とトラッキング機能の安定性の向上を実現する。

- また、上記のアルゴリズムを組み込んだシステムを開発することで、3D 都市モデルが整備されている地域であれば、現地訪問し撮影やスキャンを必要とする従来の VPS マップの構築手法と比較して、安価かつ効率的な VPS マップを作成することが出来る。

2-2. 実証フロー

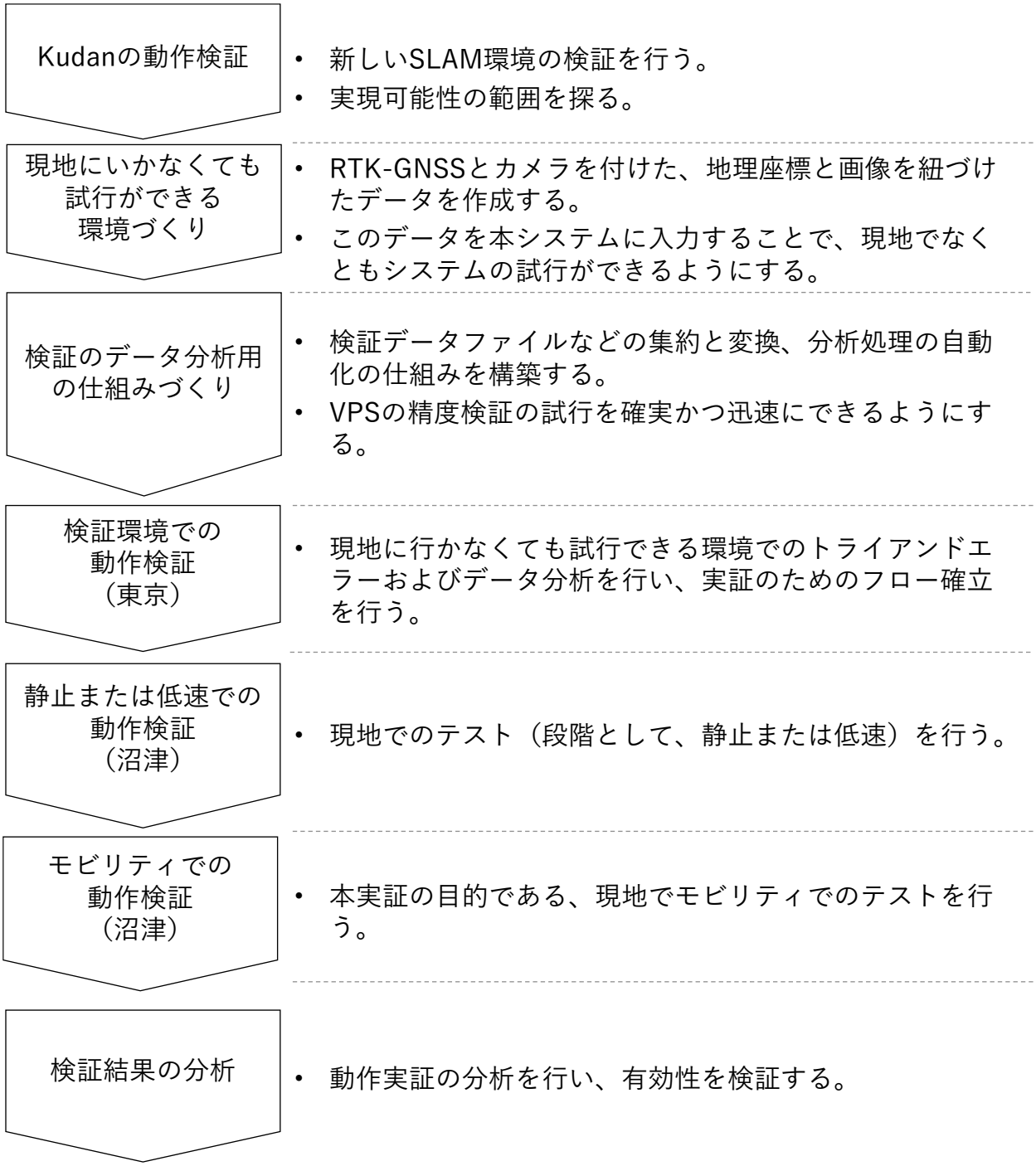


図 2-1 実証フロー

2-3. 検証ポイント

- 「KdVisual」を活用した自己位置推定アルゴリズムの機能確認
  - 今年度の検証で新たに使用する「Kdvisual」がゲームエンジン上に読み込んだ 3D 都市モデルレンダリングした画像をもとに VPS マップを作成できるかを確認する。
  - また、VPS マップ作成に当たって、仮想空間内で SLAM を利用して取得する三次元点群データが 3D 都市モデルと比較して歪みが少なく、対象ルート沿線上の建物の概形が特徴点配置として反映されているかを確認することで本アルゴリズムの有用性を確認する。
- 「C*」と「KdVisual」を使用した VPS アルゴリズムによる自己位置推定機能の向上
  - 昨年度の実証結果を踏まえ、「C*」だけでは社会実装に向けて課題となった自己位置推定の初期位置精度や継続性、処理速度に対して、3 D 都市モデルを活用した VPS アルゴリズムを 2 つ使用し、双方のアルゴリズムの処理結果から最適な座標を算出する機能を活用した本システムの有用性を確認する。

上記 2 点の検証ポイントについては、【4 章：実証技術の検証】で検証結果を記載

- 今回の実証実験で開発した車両向け自律走行システムとしての有用性検証
  - 今回開発したシステムが VPS マップ作成に必要なコスト及び準備時間に対する削減効果や、将来的な実運用を想定して安定した連続動作継続時間が担保されているのかを確認する。

上記 1 点の検証ポイントについては、【5 章：BtoB ビジネスでの有用性検証】で検証結果を記載

2-4. 実施体制

表 2-1 実施体制

役割	主体	詳細
全体管理	国土交通省 都市局	プロジェクト全体ディレクション
	アクセンチュア	プロジェクト全体マネジメント
実施事業者	ホロラボ	ユースケース実証における企画・開発・検証・運営
実施協力	産業技術総合研究所	SLAM ライブラリ「C*」提供
	Kudan	SLAM ライブラリ「KdVisual」提供
	静岡県	実証場所提供
	名古屋大学	自動運転車両提供
	東急	自動運転車両提供

2-5. 実証エリア

表 2-2 実証エリア

項目	内容
実証地	静岡県沼津市
距離(ルート)	約 2km
マップ（対象エリアは赤枠内）	

2-6. スケジュール

表 2-3 スケジュール

実施事項	2023 年									2024 年		
	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月	1 月	2 月	3 月
1. 「C*」を利用した自己位置推定機能	↔											
2.「KdVisual」を利用した自己位置推定機能	←	—	—	→								
3. 1、2 の推定位置を連携する機能			←	—	—	—	→					
4. 初期位置入力機能				←	—	—	→					
5. Status Display						←	→					
6. 各種検証					←	—	—	—	—	—	→	
7. 成果取りまとめ										←	—	→

3. 車両向け自律走行システム：実証システム

3-1. アーキテクチャ

3-1-1. システムアーキテクチャ

本システムは、昨年度「C*」をベースに開発したシステムに「KdVisual」という新たなシステムを追加することで精度、処理速度、トラッキングの継続性の面で機能高度化を目指した統合的な自己位置推定システムを開発することを目的とする。

今回新規で追加した「KdVisual」は、主に倉庫内の自動搬送ロボットやドローンへの適用を想定したカメラ画像ベースの VSLAM システムである。通常 VSLAM では実写のカメラ画像を入力し VSLAM 用のマップを作成して自己位置推定に用いるが、3D 都市モデルを活用した VPS の実現を目指す今回の実証実験では VPS マップ作成のために Unity 上で CityGML 形式の 3D 都市モデルをレンダリングした画像を入力し、VSLAM モジュールを用いて対象エリアの 3D 点群データを生成することで、三次元位置情報と特徴点情報を持った VPS 用のマップデータを作成した。走行実証時には現実世界のカメラ画像から取得された特徴点のベクトルデータを 3D 都市モデルから作成した VPS マップデータが持つ特徴点と照合し、類似性があると判定された VPS マップデータ上の特徴点の三次元位置情報をもとに現実世界のカメラ位置を算出することで自己位置推定を行った。

さらに、今回の実証実験では「C*」と「KdVisual」の両システムを同時稼働することで双方のシステムから座標を取得し、カルマンフィルタと呼ばれる 2 つ以上の位置推定システムから得られた座標軸を組み合わせて再度対象物の位置算出を行うアルゴリズムを活用することで、現在位置の高精度な推定と安定的にトラッキングが可能なシステムを構築した。

また、これらのシステムの状態を一元的に可視化するモニタリングシステムとして「StatusDisplay」を開発した。「StatusDisplay」は 3D 都市モデルから作成した三次元マップをベースに、現実世界のカメラ画像、これと比較した際のローカライズの結果、トラッキングの状況を確認できるステータスウィンドウ等を備える。具体的には、ROSSharp というライブラリを使用し、稼働している双方のシステムからの自己位置推定した結果を ROS メッセージで取得することで走行ルート上にオブジェクトとして描画する仕組みを開発した。さらに各システムから得られた座標やシステム自体の稼働状況を自己位置推定の結果のオブジェクトと併せて同じ画面上で確認可能なインターフェースとして構築し、VPS を用いた自己位置推定精度や速度をリアルタイムな評価を可能にした。
これらを活用し、昨年度同様に走行する車両に搭載したカメラからの入力画像による自己位置推定を行い、課題となった位置推定の精度向上や自律走行時における安定的なトラッキングの継続の実現を目指した。

本システムのシステム・アーキテクチャは下図の通りである。

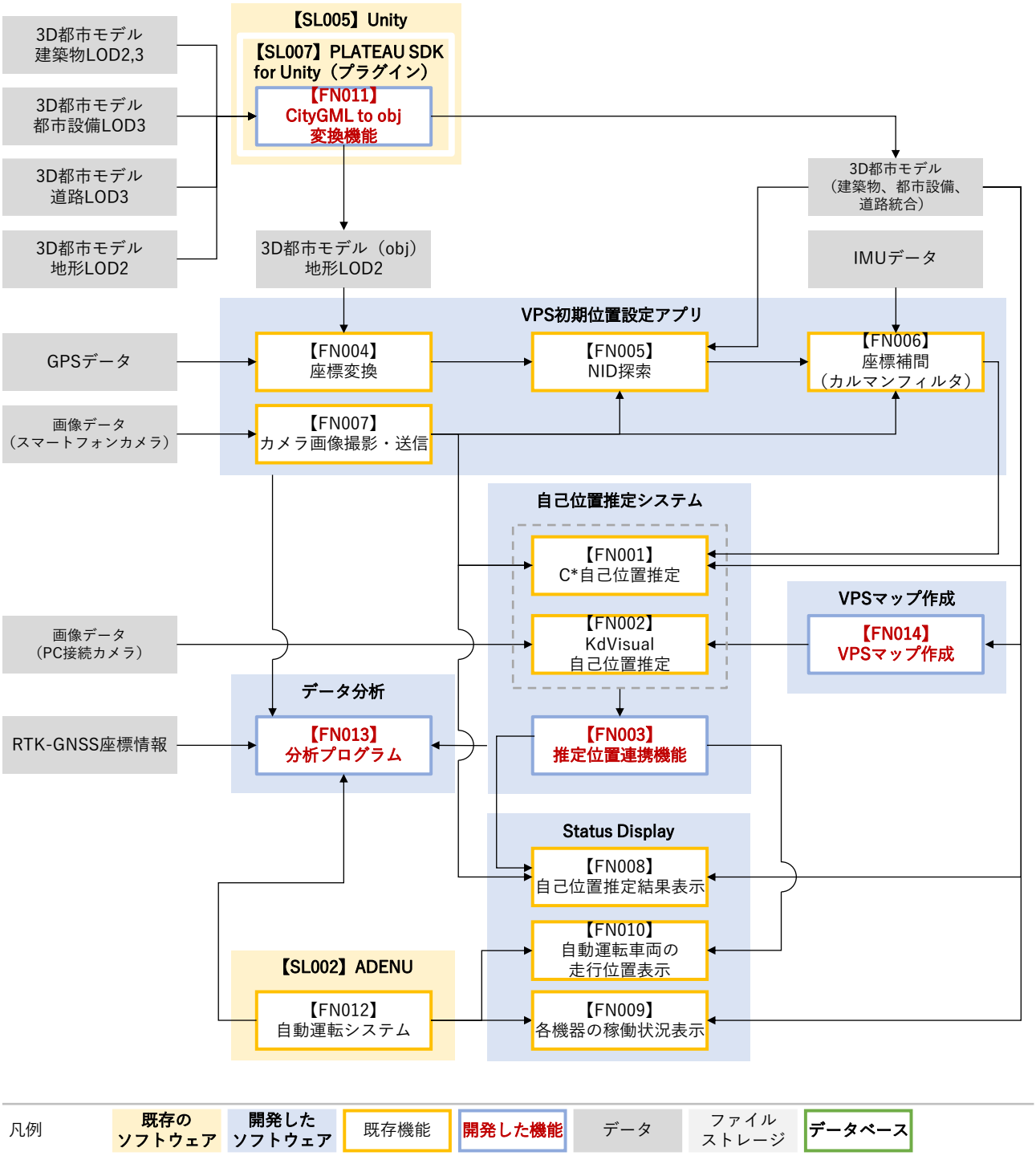


図 3-1 システムアーキテクチャ

3-1-2. データアーキテクチャ

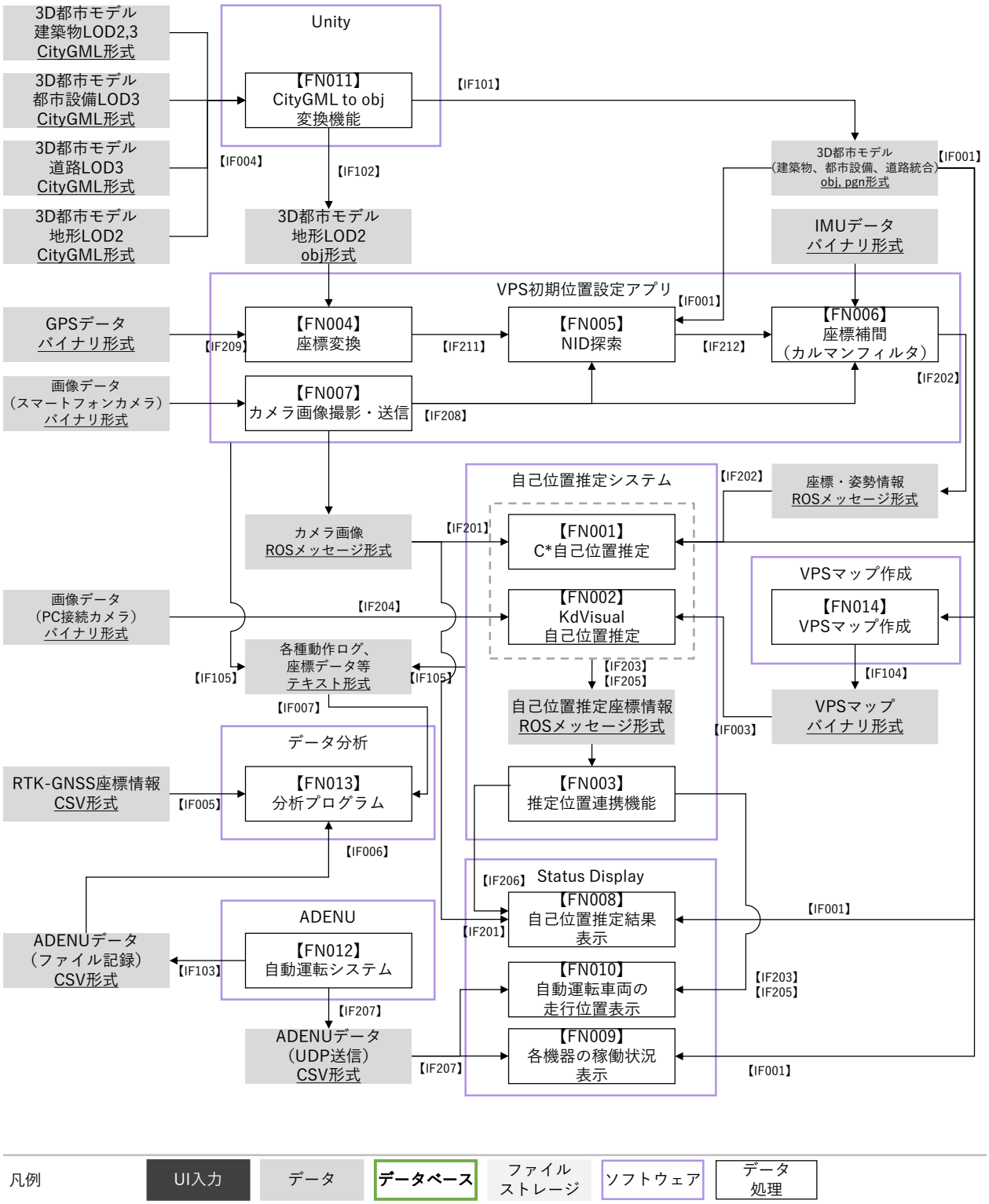


図 3-2 データアーキテクチャ

3-1-3. ハードウェアアーキテクチャ

3-1-3-a. 利用したハードウェア一覧

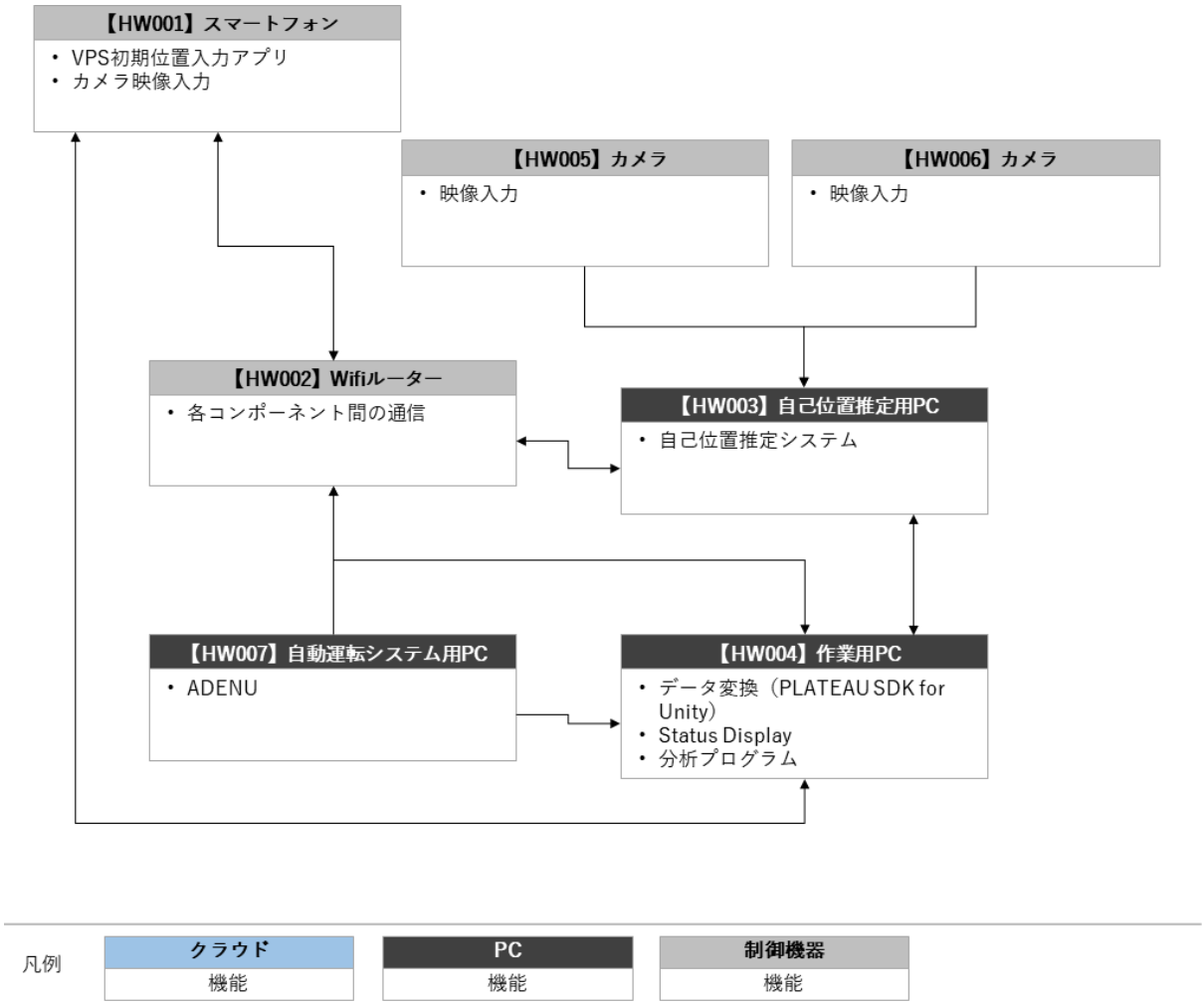


図 3-3 ハードウェアアーキテクチャ

表 3-1 利用したハードウェア一覧

ID	種別	品番	用途
HW001	スマートフォン	Google Pixel6	● GPS を利用した初期位置入力 ● カメラからの映像入力
HW002	Wifi ルーター	TP-Link Archer C55	● 各コンポーネント間の通信
HW003	PC1	OMEN by HP Laptop 16-c0161AX	● C*の動作 ● KdVisual の動作 ● 推定位置の連携 ● 各種動作ログの保存
HW004	PC2	ASUS ROG Zephyrus G14	● 各種作業用 PC、状況に合わせて様々な用途に利用 ● データ変換（PLATEAU SDK for Unity） ● Status Display の動作 ● 分析プログラム
HW005	カメラ	TIER IV C1	● 映像入力
HW006	カメラ	RealSense D455	● 映像入力
HW007	PC3	ゲーミングノート PC	● ADENU

3-1-3-b. 利用したハードウェア詳細

1)　【HW001】スマートフォン：Google Pixel 6

- 選定理由
  - 今回の実証実験に十分な処理能力とカメラ性能がある
  - 十分にコモディティ化しており、入手が容易なデバイスである
- 仕様・スペック
  - サイズ:74.8×158.6×8.9mm
  - 重量:270g
  - SoC:Google Tensor
  - RAM:8GB　ROM:128GB
  - カメラ:50M＋12M
  - センサ:GPS、加速度計、ジャイロ、磁力計など
- イメージ

図 3-4 Google Pixel 6[1]

[1] 公式 HP より抜粋： https://www.android.com/intl/ja_jp/articles/58/　（Google の端末紹介ページ）

2)　【HW002】Wifi ルーター：TP-Link Archer C55

- 選定理由
  - 今回の実証実験に必要十分な通信速度を担保することができる
- 仕様・スペック
  - 802.11 a/b/g/n/ac 対応
  - ギガビット WAN ポート
  - 867Mbps+300Mbps
- イメージ

図 3-5 TP-Link Archer C55[2]

[2] 公式 HP より抜粋：https://www.tp-link.com/jp/home-networking/wifi-router/archer-c55/

3) 【HW003】PC1：OMEN by HP Laptop 16-c0161AX

- 選定理由
  - NVIDIA® GeForce RTX™ 3070 を搭載している
  - 今回の実証実験に十分な処理能力がある
- 仕様・スペック
  - CPU:AMD Ryzen™ 7 5800H
  - GPU:NVIDIA® GeForce RTX 3070™ Laptop
  - メモリー：16GB
  - SSD：512GB
  - OS：Windows 11 Home（購入後 Ubuntu 20.04 をインストール）
- イメージ

図 3-6 OMEN by HP Laptop 16-c0161AX [3]

[3] 公式 HP より抜粋：https://jp.ext.hp.com/gaming/personal/omen_16_c/kakaku/

4) 【HW004】PC2：ASUS ROG Zephyrus G14

- 選定理由
  - 今回の実証実験に十分な処理能力がある
  - 持ち運び、実証実験中の取り回しが容易である
- 仕様・スペック
  - CPU：AMD Ryzen™ 9 5900HS
  - GPU: AMD Radeon™ ＋ NVIDIA® GeForce RTX 3060™ Laptop
  - メモリー：32GB
  - SSD:1TB
- イメージ

図 3-7 ASUS ROG Zephyrus G14[4]

[4] 公式 HP より抜粋：https://rog.asus.com/jp/laptops/rog-zephyrus/rog-zephyrus-g14-series/

5) 【HW005】カメラ：TIER IV C1

- 選定理由
  - 広いダイナミックレンジがある
  - 低ノイズで画像転送が可能である
  - LED フリッカー低減など自動運転に適した機能を有している
- 仕様・スペック
  - HDR 120dB
  - 1920x1280(2.5MP)
  - Frame Rate:Up to 30
  - Image Sensor:Sony ISX021
- イメージ

図 3-8 TIER IV C1[5]

[5] 公式 HP より抜粋：https://sensor.tier4.jp/automotive-camera/#C1

6) 【HW006】カメラ：Intel® RealSense™ Depth Camera D455

- 選定理由
  - ステレオデプスカメラとして、各種開発環境の整備が容易なため
- 仕様・スペック
  - RGB:1280x800
  - Depth:1280x720
  - Frame Rate:Up to 30
  - 124mmx26mmx29mm
- イメージ

図 3-9 Intel Realsense Depth Camera D455[6]

7) 【HW007】PC3：ゲーミングノート PC

- 選定理由
  - ADENU 制御 PC
- 仕様・スペック
  - ADENU を動作することが可能な処理能力を有している
  - NVIDIA GPU の搭載及び Ubuntu Linux の OS を使用している

[6] 公式 HP より抜粋：https://www.intelrealsense.com/depth-camera-d455/

3-2. システム機能

3-2-1. システム機能一覧

1） PC 機能一覧

表 3-2 システム機能一覧

※赤文字：既存改修・新規開発

大分類	小分類	ID	機能名	機能説明
自己位置推定機能	C*を利用した自己位置推定機能	FN001	C* 自己位置推定	● 単眼カメラの画像と初期位置の入力から 3D 都市モデルのレンダリング画像と比較し自己位置を推定する機能
	KdVisual を利用した自己位置推定機能	FN002	KdVisual 自己位置推定	● 単眼カメラの画像と初期位置の入力から 3D 都市モデルのレンダリング画像と比較し自己位置を推定する機能
自己位置推定補助機能	推定位置を連携する機能	FN003	推定位置連携機能	● FN001、FN002 の出力する座標値やセンサ値を適切に利用し、最終的な自己位置推定座標を出力する機能 ● FN001、FN002 の出力した座標を元により確からしい座標を出力する
	初期位置を入力する機能	FN004	座標変換	● スマートフォンの GPS 機能から現在地の緯度経度を取得し、内部座標系に変換する機能
		FN005	NID 探索	● GPS 座標を初期位置として周囲およそ 10m の範囲で 3D 都市モデルとカメラ画像の NID が最低の場所を探索し、その座標を出力する機能
		FN006	座標補間（カルマンフィルタ）	● 入力した座標と IMU から座標補間を行い、その座標を出力する機能
		FN007	カメラ画像撮影・送信	● スマートフォンのカメラ画像を ROS メッセージにして送信する機能
状態表示機能	システムの状態を表示する機能	FN008	自己位置推定結果表示	● 推定された自己位置にカメラを表すオブジェクトを表示する機能
		FN009	各機器の稼働状況表示	● 通信状況から各機器の稼働状況・接続状況を表示する機能
		FN010	自動運転車両の走行位置表	● 自動運転車からの走行位置を表示する機能

			示	
データ変換	CityGML to obj 変換機能	FN011	CityGML to obj 変換	● 3D 都市モデルを CityGML から各機能で利用可能な obj 形式の 3D データに変換する機能
自動運転	自動運転システム	FN012	自動運転システム	● 自動運転機能
データ分析	データを分析する機能	FN013	分析プログラム	● ログに出力されたデータを分析する機能
VPS マップ作成	VPS マップを作成する機能	FN014	VPS マップ作成	● 3D 都市モデルのレンダリング画像を ROS メッセージで送信し、KdVisual の SLAM モジュールに入力する機能

3-2-2. 利用したソフトウェア・ライブラリ

表 3-3 利用したソフトウェア・ライブラリ

※赤文字：既存改修・新規開発

ID	項目	内容
SL001	C*	● 産業技術総合研究所で研究・開発された単眼カメラによる VPS ソフトウェア
SL002	ADENU	● 名古屋大学 COI で研究・開発された自動運転車用のソフトウェアパッケージ
SL003	KdVisual	● Kudan 社が開発した SLAM ソフトウェア
SL004	ROS	● Robot Operating System の略 ● ロボット・アプリケーション作成を支援するライブラリとツール
SL005	Unity	● Unity 社が提供する、ゲームエンジン
SL006	ROS#	● C#で実装された ROS コネクタ ● Unity から ROS に接続するために利用
SL007	PLATEAU SDK for Unity[7]	● PLATEAU の 3D 都市モデルデータを Unity で扱うためのツールキット

[7] Synesthesias/PLATEAU-SDK-for-Unity: PLATEAU の 3D 都市モデルデータを Unity で扱うためのツールキット（github.com）

3-2-3. 開発機能の詳細要件

1) システム機能一覧

1. 【FN001】C*自己位置推定

- 機能概要
  - 単眼カメラ画像と 3D 都市モデルのレンダリング画像を対照させカメラの位置を推定する機能
- フローチャート

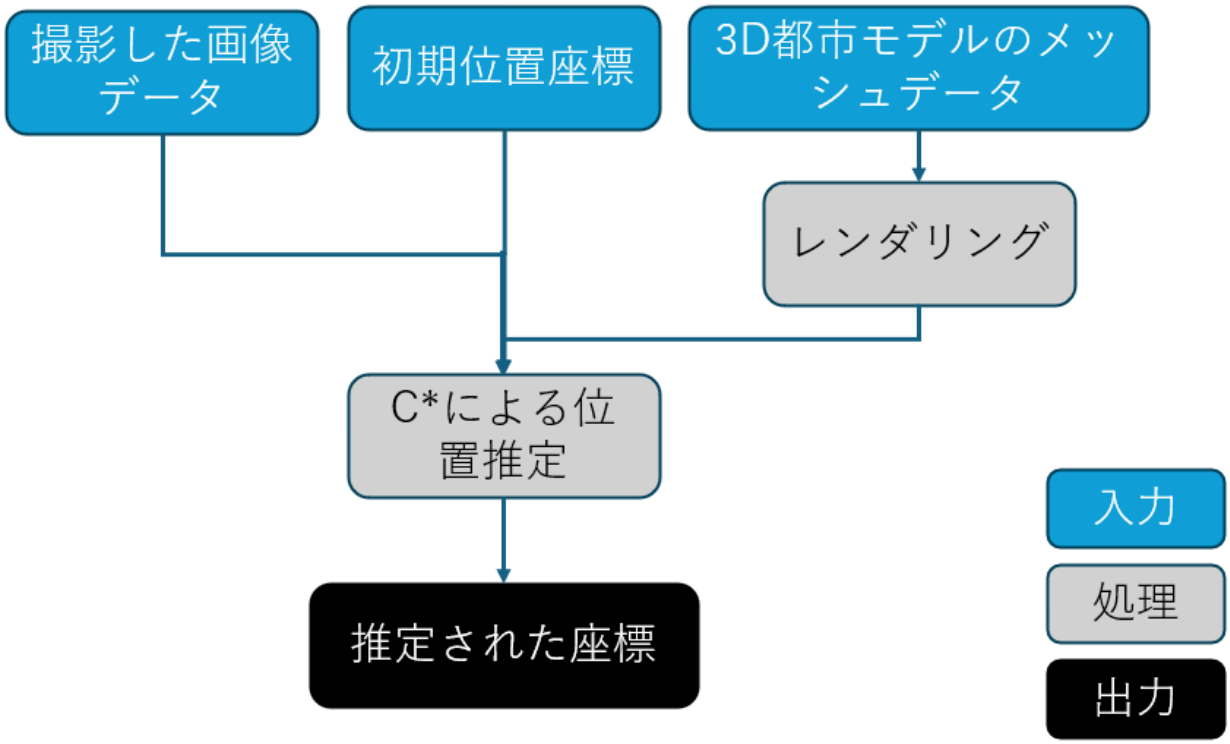


図 3-10 C*のフローチャート

- データ仕様
  - 入力
    - 画像データ
      - 内容
        - スマートフォンまたはカメラからの画像
      - 形式
        - 無圧縮、または JPEG 圧縮されたバイト列を内包した CompressedImage 形式の ROS メッセージ
      - データ詳細
        - 内部連携インターフェース【IF201】を参照
    - 初期位置入力の座標データ
      - 内容

        - カメラ画像データとカメラ姿勢の位置・回転の数値データ
      - 形式
        - 無圧縮、または JPEG 圧縮されたバイト列と、位置・回転ベクトルの数値データを内包した、PosedImage 形式の ROS メッセージ
      - データ詳細
        - 内部連携インターフェース【IF202】を参照
    - 3D 都市モデルのテクスチャ付き 3D メッシュデータ
      - 内容
        - 3D 都市モデルのメッシュデータ
      - 形式
        - CityGML から OBJ 形式に変換したファイル
      - データ詳細
        - ファイル入力インターフェース【IF001】を参照
  - 出力
    - 推定された位置情報
      - 内容
        - 推定されたカメラ姿勢の位置・回転の数値データ
        - 座標系は地理座標と相互変換可能な内部座標系
      - 形式
        - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
      - データ詳細
        - 内部連携インターフェース【IF203】を参照
- 機能詳細
  - 画像レンダリング
    - 処理内容
      - 3D 都市モデルをレンダリングし、対照用のカラー画像と深度画像を得る
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL005】を参照）
      - ROS#（ソフトウェア・ライブラリ【SL006】を参照）
    - 利用するアルゴリズム
      - なし
  - 位置推定
    - 処理内容
      - 初期位置入力がある場合は、その位置でのレンダリング画像を作成
      - ない場合は、C*の調整（アルゴリズム【AL102】を参照）により推測された位置と直前位置の比較を行い、距離が閾値を超えていたらレンダリング画像を生成
      - 入力画像とレンダリング画像の NID（アルゴリズム【AL002】を参照）を計算
      - レンダリング画像の再投影を行いカメラ視点を疑似的に変更し、BFGS 法（アルゴリズム

【AL03】を参照）で NID が最小値になるカメラ視点を探索する

- ✧ 利用するライブラリ
  - C*（ソフトウェア・ライブラリ【SL001】を参照）
  - ROS（ソフトウェア・ライブラリ【SL004】を参照）
- ✧ 利用するアルゴリズム
  - C*（アルゴリズム【AL001】を参照）
  - NID（アルゴリズム【AL002】を参照）
  - BFGS 法（アルゴリズム【AL03】を参照）
  - C*の調整（アルゴリズム【AL102】を参照）

2. 【FN002】KdVisual 自己位置推定

- 機能概要
  - ➢ 単眼カメラまたはステレオカメラ画像と、あらかじめ構築しておいたマップデータからカメラ位置を推定する機能
- フローチャート

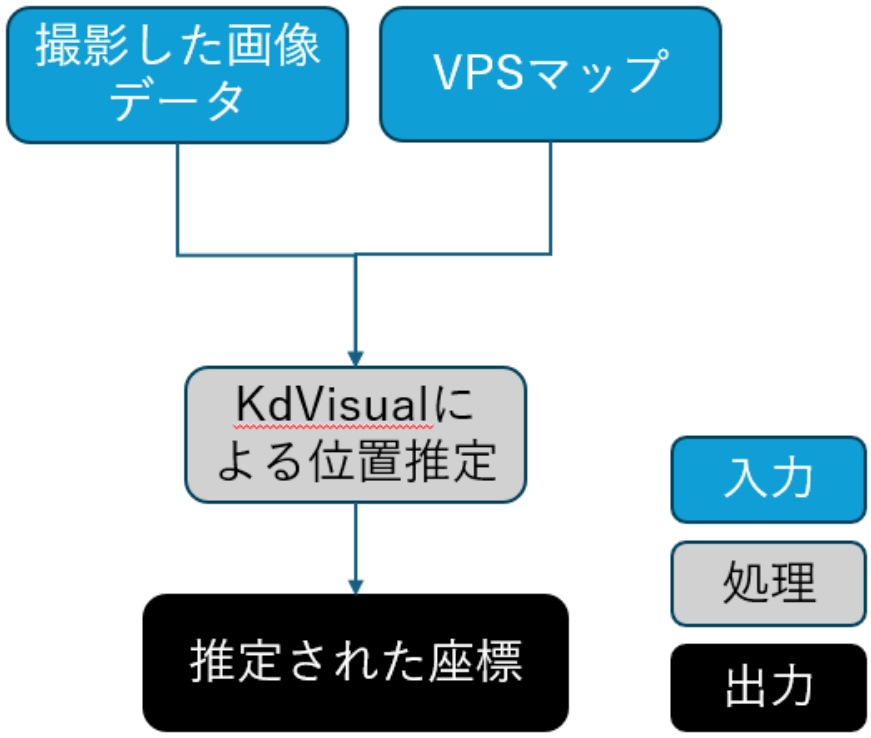


図 3-11 KdVisual のフローチャート

- データ仕様
  - ➢ 入力
    - ✧ 画像データ
      - 内容
        - ➢ カメラからの画像データ
      - 形式

- 無圧縮、または JPEG 圧縮されたバイト列を内包した CompressedImage 形式の ROS メッセージ
- データ詳細
  - 内部連携インターフェース【IF204】を参照
- マップデータ
  - 内容
    - あらかじめ構築したマップデータ
  - 形式
    - KdVisual の独自バイナリ形式
  - データ詳細
    - ファイル入力インターフェース【IF003】を参照
- 出力
  - 推定された位置情報
    - 内容
      - 推定されたカメラ姿勢の位置・回転の数値データ
      - 座標系は KdVisual が認識している独自のローカル座標系
    - 形式
      - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
    - データ詳細
      - 内部連携インターフェース【IF205】を参照

- 機能詳細
  - 位置推定(VPS)
    - 処理内容
      - VisualSLAM 処理のうち、マップの作成を行わず既存のマップの照合のみ行い自己位置を推定するモード
      - KdVisual の tracking_only オプションを有効にすることで動作する
      - 入力されたカメラ画像の特徴点と、マップに格納されている特徴点を比較・照合し、自己位置を推定する
    - 利用するライブラリ
      - KdVisual（ソフトウェア・ライブラリ【SL003】を参照）
      - ROS（ソフトウェア・ライブラリ【SL004】を参照）
    - 利用するアルゴリズム
      - VisualSLAM（アルゴリズム【AL004】を参照）
  - VisualSLAM
    - 処理内容
      - 入力されたカメラ画像を元に、自己位置の推定と周囲のマップの作成を行う
      - 入力された画像の特徴点とその特徴量を計算し、自己位置推定や周辺環境の三次元再構築を行う

- ✧ 利用するライブラリ
  - KdVisual（ソフトウェア・ライブラリ【SL003】を参照）
  - ROS（ソフトウェア・ライブラリ【SL004】を参照）
- ✧ 利用するアルゴリズム
  - VisualSLAM（アルゴリズム【AL004】を参照）

3. 【FN003】推定位置連携機能

- 機能概要
  - C*（機能【FN001】を参照）と KdVisual（機能【FN002】を参照）が出力する座標から、確からしい座標を出力する機能
- フローチャート

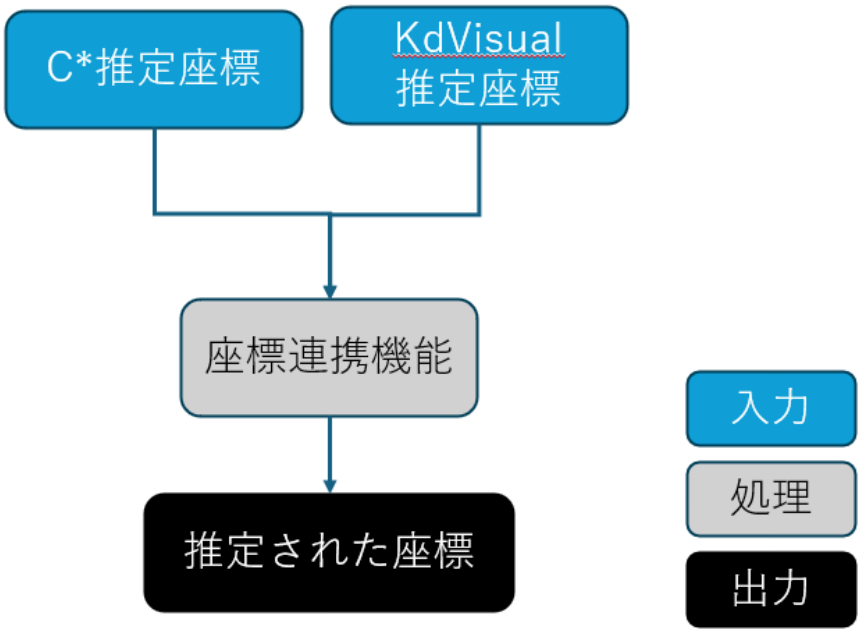


図 3-12 座標連携機能のフローチャート

- データ仕様
  - 入力
    - ✧ 推定された位置情報
      - 内容
        - 推定されたカメラ姿勢の位置・回転の数値データ
        - 座標系は KdVisual が認識している独自のローカル座標系
      - 形式
        - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
      - データ詳細
        - 内部連携インターフェース【IF203】【IF205】を参照
  - 出力

- ✧ 推定された位置情報
  - 内容
    - ➢ 推定されたカメラ姿勢の位置・回転の数値データ
    - ➢ 座標系は地理座標と相互変換可能な内部座標系
  - 形式
    - ➢ 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
  - データ詳細
    - ➢ 内部連携インターフェース【IF206】を参照

- 機能詳細
  - ➢ 座標連携機能
    - ✧ 処理内容
      - C*（機能【FN001】を参照）と KdVisual（機能【FN002】を参照）により推定された座標とセンサ情報などを組み合わせて、より確からしい推定座標を出力する
    - ✧ 利用するライブラリ
      - ROS（ソフトウェア・ライブラリ【SL004】を参照）
    - ✧ 利用するアルゴリズム
      - 推定位置連携アルゴリズム（アルゴリズム【AL101】を参照）

4. 【FN004】座標変換

- 機能概要
  - スマートフォンの GPS 機能から現在地の緯度経度を取得し、内部座標系に変換する機能
- フローチャート

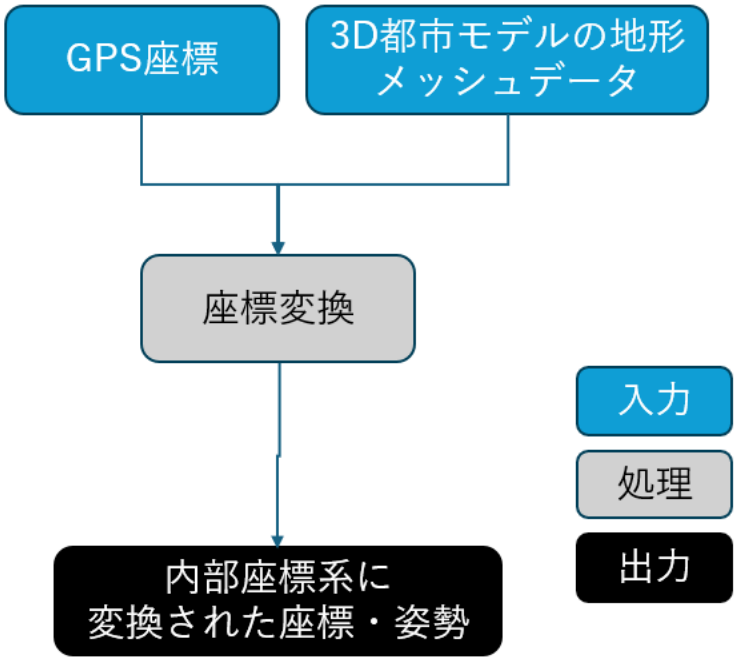


図 3-13 座標変換機能のフローチャート

- データ仕様
  - 入力
    - スマートフォン GPS 情報
      - 内容
        - スマートフォン搭載の GPS により取得される位置情報
      - 形式
        - 緯度経度をそれぞれ Double 型で取得
      - データ詳細
        - Double 型数値
        - スマートフォンの API から取得
        - 内部連携インターフェース【IF209】を参照
    - 3D 都市モデルの地形 3D メッシュデータ
      - 内容
        - 3D 都市モデルの地形メッシュデータ
      - 形式
        - CityGML から OBJ 形式に変換したファイル
      - データ詳細
        - ファイル入力インターフェース【IF002】を参照

- 出力
  - 座標データ
    - 内容
      - 内部座標系に変換された座標値と姿勢
    - 形式
      - Vector3 型の数値
      - Quaternion 型の数値
    - データ詳細
      - 位置と回転を表す数値
      - 内部連携インターフェース【IF211】を参照

- 機能詳細
  - 座標変換
    - 処理内容
      - スマートフォンの GPS 座標を API から取得し、平面直角座標 8 系への変換と実証エリアに設定した原点へのオフセット計算を行う。
      - 高さについては、変換した座標の二次元座標の場所の 3D 都市モデルの地形データを参照し、地表から設定した距離離れた場所を計算し出力する。
      - Mesh Collider を設定した地形データに対して、現在座標から鉛直下向きに Raycast を行い、衝突座標の Y 座標を地面の高さとして設定した定数値を加算したものを現在座標の高さとして用いる。
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL005】を参照）
    - 利用するアルゴリズム
      - なし

5. 【FN005】NID 探索

- 機能概要
  - スマートフォン GPS 座標を初期位置として、周辺の複数位置でのカメラと 3D 都市モデルのレンダリング画像の NID を計算し、最小値の座標を出力する。
- フローチャート

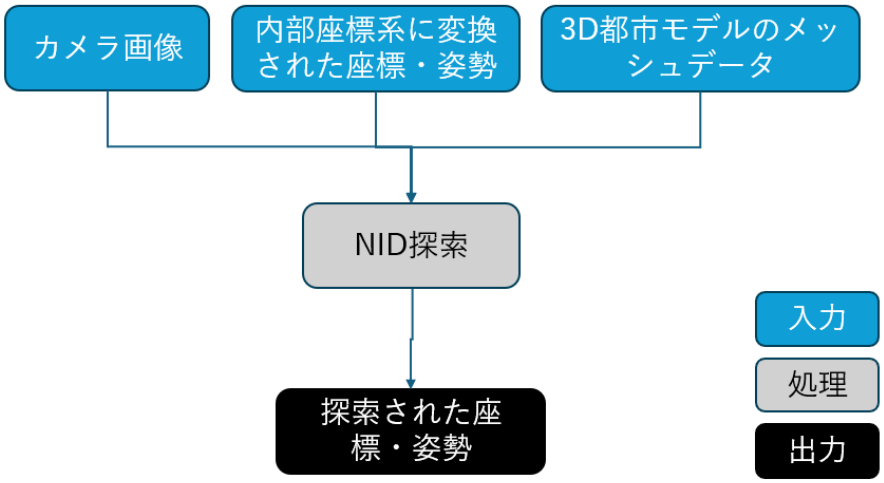


図 3-14 NID 探索機能のフローチャート

- データ仕様
  - 入力
    - スマートフォンカメラ画像
      - 内容
        - スマートフォンカメラが撮影している画像
      - 形式
        - バイナリ配列
      - データ詳細
        - プログラム内で直接取得するカメラ画素の色情報を表す RGB のデータ配列
        - 内部連携インターフェース【IF208】を参照
    - 【FN004】により変換された座標・姿勢
      - 内容
        - スマートフォン搭載の GNSS により取得される位置情報を内部座標系に変換したもの
      - 形式
        - Vector3 型の数値
        - Quaternion 型の数値
      - データ詳細
        - 位置と回転を表す数値
        - 内部連携インターフェース【IF211】を参照
    - 3D 都市モデルのテクスチャ付き 3D メッシュデータ

- 内容
  - 3D 都市モデルのメッシュデータ
- 形式
  - 【FN011】により CityGML から OBJ 形式に変換したファイル
- データ詳細
  - ファイル入力インターフェース【IF001】を参照

➢ 出力

✧ 探索結果の座標・姿勢

- 内容
  - 探索結果の、範囲内で NID が最小になるポイントの座標・姿勢
- 形式
  - Vector3 型の数値
  - Quaternion 型の数値
- データ詳細
  - 位置と回転を表す数値
  - 内部連携インターフェース【IF212】を参照

- 機能詳細

➢ NID 探索機能

✧ 処理内容

- 【FN004】によりスマートフォン GPS 座標から変換された座標を原点とするおよそ 10m の範囲で複数の Camera コンポーネントをあらかじめ配置した GameObject を配置する。
- Camera コンポーネントでレンダリングした画像は 320x240 の画像が 16 枚まとめられた RenderTexture に格納される。
- この RenderTexture に対して Compute Shader で作成したプログラムでカメラ画像との NID を計算し、NID が最小である Camera を特定し、その座標を出力する。

✧ 利用するライブラリ

- Unity（ソフトウェア・ライブラリ【SL005】を参照）

✧ 利用するアルゴリズム

- NID（アルゴリズム【AL002】を参照）

6. 【FN006】座標補間

- 機能概要
  - 入力座標と IMU から座標補間を行い、その座標を出力する機能。カメラ画像も取り込み、座標付きの画像として出力する。
  - この出力が初期位置入力となる。
- フローチャート

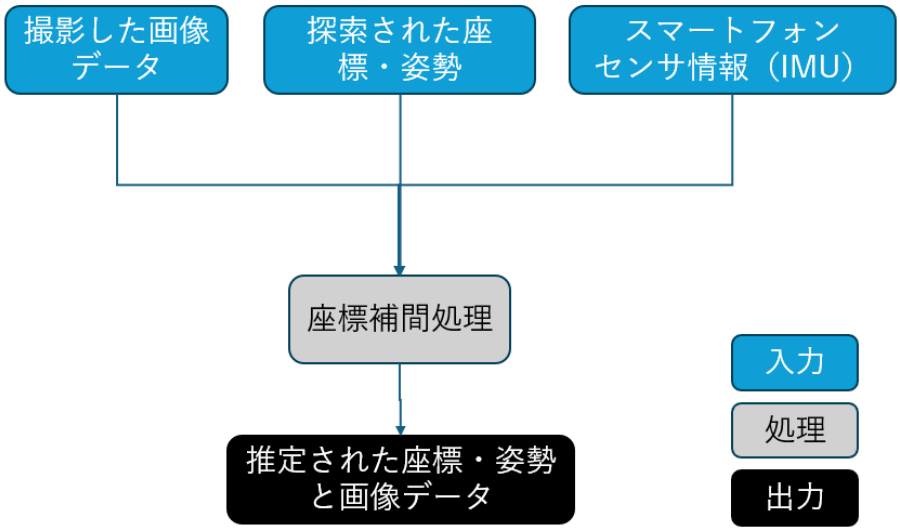


図 3-15 座標補間機能のフローチャート

- データ仕様
  - 入力
    - ✧ 探索された座標・姿勢
      - 内容
        - 【FN005】NID 探索機能の結果の座標
      - 形式
        - Vector3 型の数値
        - Quaternion 型の数値
      - データ詳細
        - 位置と回転を表す数値
        - 内部連携インターフェース【IF212】を参照
    - ✧ スマートフォン IMU 情報
      - 内容
        - スマートフォン搭載のセンサにより取得される各種センサ値
      - 形式
        - 複数の Vector3 型の数値
      - データ詳細
        - 加速度、角速度のベクトルを表す数値
        - 内部連携インターフェース【IF210】を参照

- ✧ スマートフォンカメラ画像
  - 内容
    - スマートフォンカメラが撮影している画像
  - 形式
    - バイナリ配列
  - データ詳細
    - プログラム内で直接取得するカメラ画素の色情報を表す RGB のデータ配列
    - 内部連携インターフェース【IF208】を参照
- 出力
  - ✧ 推定された座標・姿勢と画像データ
    - 内容
      - カメラ画像データとカメラの座標・姿勢の数値データ
    - 形式
      - 無圧縮、または JPEG 圧縮されたバイト列と、位置・回転ベクトルの数値データを内包した、PosedImage 形式の ROS メッセージ
    - データ詳細
      - PosedImage 形式の ROS メッセージとして補間処理によって推定された座標と姿勢をカメラ画像と合わせて C*の自己位置推定のための初期位置として出力する
      - 内部連携インターフェース【IF202】を参照

- 機能詳細
  - 座標補間処理
    - ✧ 処理内容
      - カルマンフィルタ【AL005】により、スマートフォンセンサ情報と【FN005】の NID 探索の結果座標から、より確からしい現在位置を推定する
      - 推定した位置とスマートフォンのカメラ画像を合わせて、ROS メッセージとして C*に初期位置を入力する
    - ✧ 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL005】を参照）
      - ROS#（ソフトウェア・ライブラリ【SL006】を参照）
    - ✧ 利用するアルゴリズム
      - カルマンフィルタ（アルゴリズム【AL005】を参照）

7. 【FN007】カメラ画像撮影・送信

- 機能概要
  - スマートフォンのカメラ画像を ROS メッセージにして送信する機能
- フローチャート

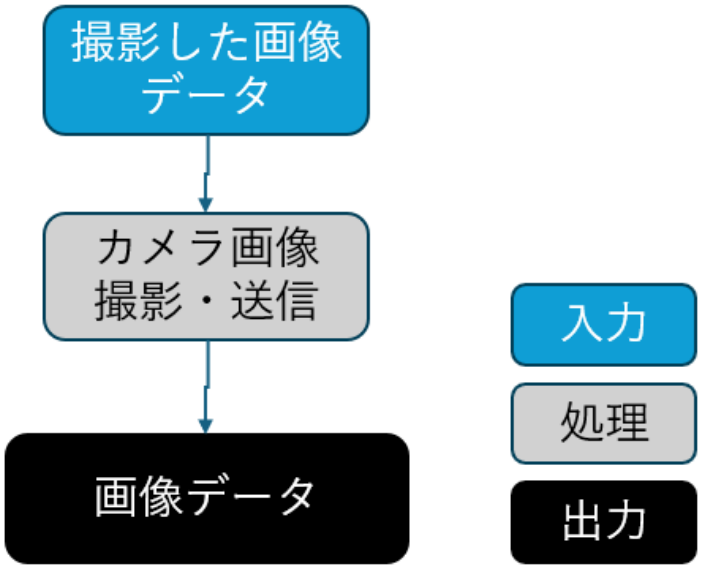


図 3-16 カメラ画像撮影・送信機能のフローチャート

- データ仕様
  - 入力
    - スマートフォンカメラ画像
      - 内容
        - スマートフォンカメラが撮影している画像
      - 形式
        - バイナリ配列
      - データ詳細
        - プログラム内で直接取得するカメラ画素の色情報を表す RGB のデータ配列
        - 内部連携インターフェース【IF208】を参照
  - 出力
    - 画像データ
      - 内容
        - スマートフォンまたはカメラからの画像データ
      - 形式
        - 無圧縮、または JPEG 圧縮されたバイト列を内包した CompressedImage 形式の ROS メッセージ
      - データ詳細
        - 内部連携インターフェース【IF201】を参照

- 機能詳細
  - カメラ画像撮影・送信機能
    - 処理内容
      - スマートフォンのカメラ画像を ROS メッセージに変換し送信する
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL005】を参照）
      - ROS#（ソフトウェア・ライブラリ【SL006】を参照）
    - 利用するアルゴリズム
      - なし

8. 【FN008】自己位置推定結果表示

- 機能概要
  - 自己位置推定結果の座標にオブジェクトを配置し、3D でグラフィカルに表示する
- フローチャート

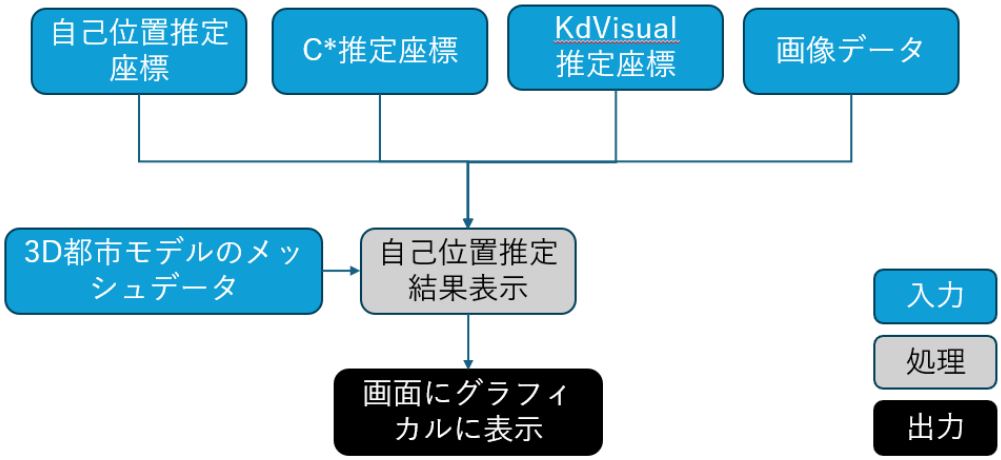


図 3-17 自己位置推定結果表示機能のフローチャート

- データ仕様
  - 入力
    - 自己位置推定座標
      - 内容
        - 推定されたカメラの座標・姿勢の数値データ
        - 座標系は地理座標と相互変換可能な内部座標系
      - 形式
        - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
      - データ詳細
        - 【FN003】により推定された座標
        - 内部連携インターフェース【IF206】を参照

- ✧ KdVisual により推定された位置情報
  - 内容
    - 推定されたカメラの座標・姿勢の数値データ
    - 座標系は KdVisual が認識している独自のローカル座標系
  - 形式
    - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
  - データ詳細
    - 【FN002】により推定された座標
    - 内部連携インターフェース【IF205】を参照
- ✧ C*により推定された位置情報
  - 内容
    - 推定されたカメラの座標・姿勢の数値データ
    - 座標系は地理座標と相互変換可能な内部座標系
  - 形式
    - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
  - データ詳細
    - 【FN001】により推定された座標
    - 内部連携インターフェース【IF203】を参照
- ✧ カメラ画像
  - 内容
    - スマートフォンのカメラ画像
  - 形式
    - 無圧縮、または JPEG 圧縮されたバイト列を内包した CompressedImage 形式の ROS メッセージ
  - データ詳細
    - 内部連携インターフェース【IF201】を参照
- ✧ 3D 都市モデルのテクスチャ付き 3D メッシュデータ
  - 内容
    - 3D 都市モデルのメッシュデータ
  - 形式
    - 【FN011】により CityGML から OBJ 形式に変換したファイル
  - データ詳細
    - ファイル入力インターフェース【IF001】を参照

➢ 出力

- ✧ なし（画面表示）

- 機能詳細
  - ➢ 状況表示機能
    - ✧ 処理内容

- 各機能から入力された情報を、画面に表示する
- 各機能の位置を表す GameObject の Transform に入力された位置と姿勢の情報を反映させる
- 背景として 3D 都市モデルを描画し、推定位置をわかりやすく表示する

✧ 利用するライブラリ

- Unity（ソフトウェア・ライブラリ【SL005】を参照）
- ROS#（ソフトウェア・ライブラリ【SL006】を参照）

✧ 利用するアルゴリズム

- なし

9. 【FN009】各機器の稼働状況表示

- 機能概要
  - ➢ 通信状況から各機器の稼働状況・接続状況を表示する機能
- フローチャート

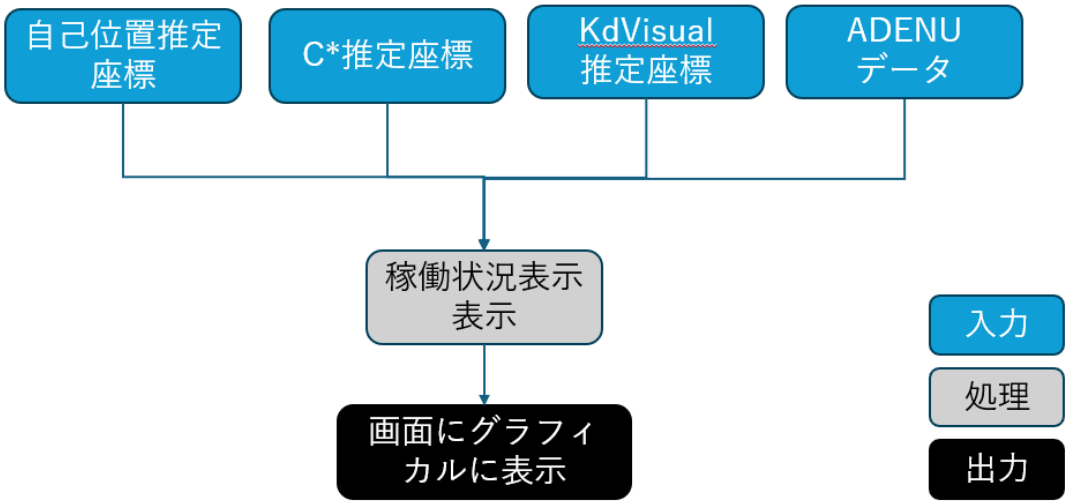


図 3-18 各機器の稼働状況表示機能のフローチャート

- データ仕様
  - ➢ 入力
    - ✧ 自己位置推定座標
      - 内容
        - ➢ 推定されたカメラの座標・姿勢の数値データ
        - ➢ 座標系は地理座標と相互変換可能な内部座標系
      - 形式
        - ➢ 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
      - データ詳細
        - ➢ 【FN002】により推定された座標
        - ➢ 内部連携インターフェース【IF206】を参照
    - ✧ KdVisual により推定された位置情報

- 内容
  - 推定されたカメラの座標・姿勢の数値データ
  - 座標系は KdVisual が認識している独自のローカル座標系
- 形式
  - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
- データ詳細
  - 【FN002】により推定された座標
  - 内部連携インターフェース【IF205】を参照

✧ C*により推定された位置情報

- 内容
  - 推定されたカメラの座標・姿勢の数値データ
  - 座標系は地理座標と相互変換可能な内部座標系
- 形式
  - 位置・回転ベクトルの数値データを内包した、TF 形式の ROS メッセージ
- データ詳細
  - 【FN001】により推定された座標
  - 内部連携インターフェース【IF203】を参照

✧ 自動運転車の位置情報

- 内容
  - 自動運転車が認識している座標・姿勢の数値データ
  - 座標系は平面直角座標 8 系
- 形式
  - 位置・回転ベクトルの数値データを CSV で表した文字列を UDP 通信で受信する
- データ詳細
  - 内部連携インターフェース【IF207】を参照

➢ 出力

✧ なし（画面表示）

- 機能詳細

➢ 各機器の稼働状況表示機能

✧ 処理内容

- データ受信により死活監視、接続状況の確認を行う
- 通信が成立しデータを受信していることを持って稼働状態と判断する

✧ 利用するライブラリ

- ADENU（ソフトウェア・ライブラリ【SL002】を参照）
- Unity（ソフトウェア・ライブラリ【SL005】を参照）
- ROS#（ソフトウェア・ライブラリ【SL006】を参照）

✧ 利用するアルゴリズム

- なし

10. 【FN010】自動運転車両の走行位置表示

- 機能概要
  - 自動運転車両の走行位置をグラフィカルに表示する
- フローチャート

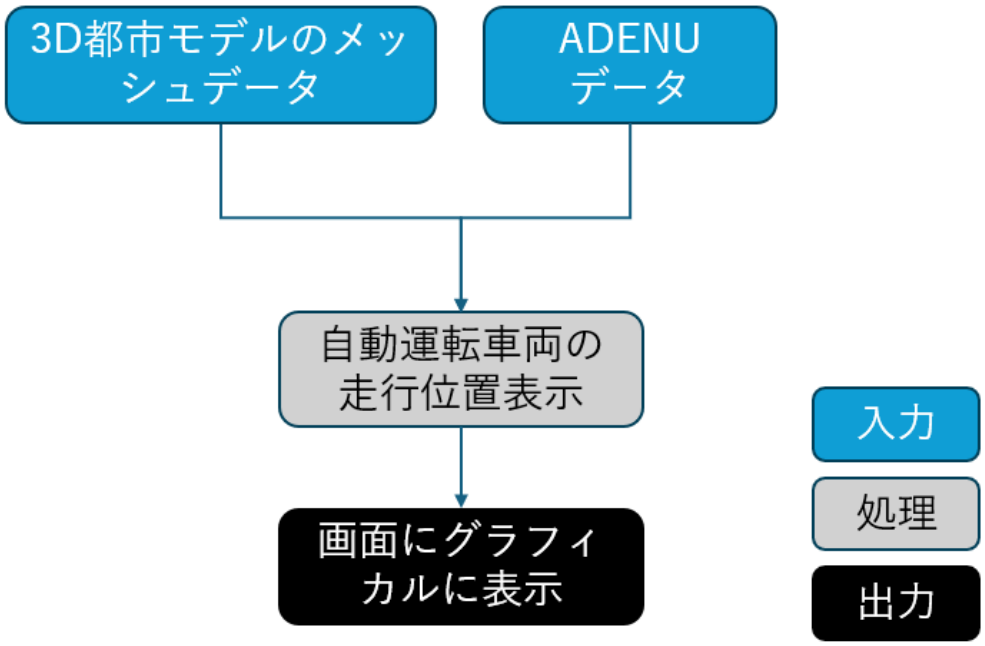


図 3-19 自動運転車両の走行位置表示機能のフローチャート

- データ仕様
  - 入力
    - 自動運転車の位置情報
      - 内容
        - 自動運転車が認識している座標・姿勢の数値データ
        - 座標系は平面直角座標 8 系
      - 形式
        - 位置・回転ベクトルの数値データを CSV で表した文字列を UDP 通信で受信する
      - データ詳細
        - 内部連携インターフェース【IF207】を参照
    - 3D 都市モデルのテクスチャ付き 3D メッシュデータ
      - 内容
        - 3D 都市モデルのメッシュデータ
      - 形式
        - 【FN011】により CityGML から OBJ 形式に変換したファイル
      - データ詳細
        - ファイル入力インターフェース【IF001】を参照
  - 出力

- ✧ なし（画面表示）

- 機能詳細
  - ➢ 自動運転車両の走行位置表示機能
    - ✧ 処理内容
      - 自動運転車のシステムから入力された情報を、画面に表示する
      - 平面直角座標 8 系で入力される座標を内部座標に変換する
        - ➢ 検証エリアの原点でオフセット計算を行う
    - ✧ 利用するライブラリ
      - ADENU（ソフトウェア・ライブラリ【SL002】を参照）
      - Unity（ソフトウェア・ライブラリ【SL005】を参照）
    - ✧ 利用するアルゴリズム
      - なし

11. 【FN011】CityGML to obj 変換

- 機能概要
  - ➢ CityGML の 3D 都市モデルを各機能で利用可能な obj 形式の 3D データに変換する機能
- フローチャート

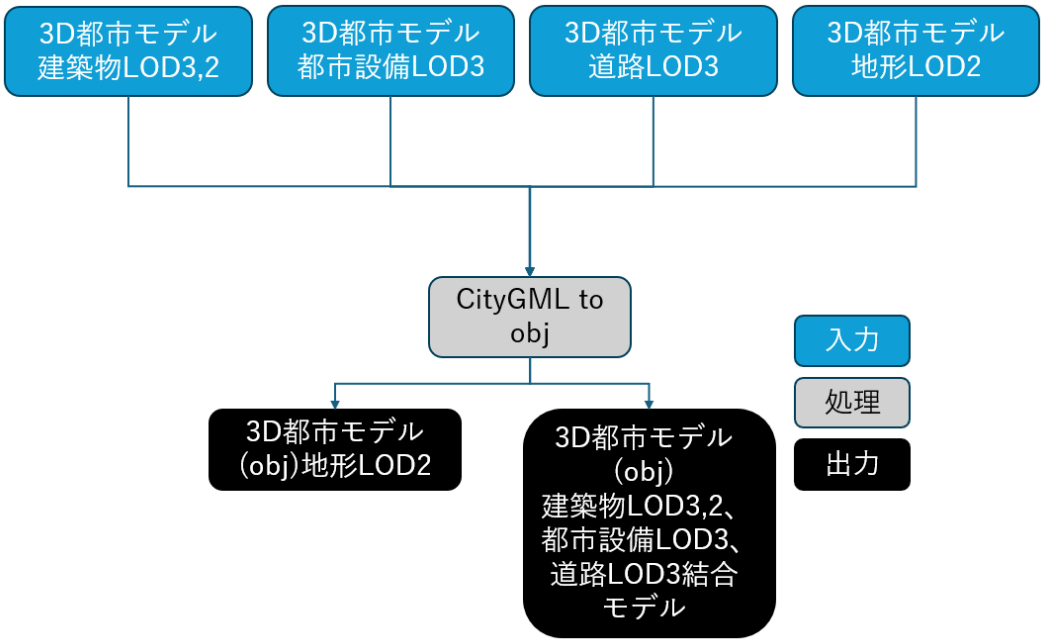


図 3-20 CityGML to obj 変換機能のフローチャート

- データ仕様
  - ➢ 入力
    - ✧ 3D 都市モデル
      - 内容
        - ➢ 3D 都市モデルの CityGML ファイル
      - 形式

        - CityGML
    - データ詳細
        - 使用した地物型
            - 建築物 LOD3
            - 建築物 LOD2（LOD3 がない部分で利用）
            - 都市設備 LOD3
            - 道路 LOD3
            - 地形 LOD2
        - ファイル入力インターフェース【IF004】を参照
- 出力
    - 3D 都市モデル(obj)
        - 内容
            - 3D 都市モデルの obj ファイルとテクスチャの PNG ファイル
        - 形式
            - obj 形式ファイル
            - PNG 形式ファイル
        - データ詳細
            - 対象エリアの 3D 都市モデルを利用しやすいように obj 形式に変換したもの
            - 扱いやすいように 10 エリアに分割
            - ファイル出力インターフェース【IF101】【IF102】を参照
- 機能詳細
    - CityGML to obj 変換機能
        - 処理内容
            - 3D 都市モデルの CityGML を読み込み、3D モデルを obj 形式で出力する
            - PLATEAU SDK for Unity を利用、分割や obj 出力はマニュアル操作で対応
        - 利用するライブラリ
            - Unity（ソフトウェア・ライブラリ【SL005】を参照）
            - PLATEAU SDK for Unity（ソフトウェア・ライブラリ【SL007】を参照）
        - 利用するアルゴリズム
            - なし

12. 【FN012】自動運転システム

- 機能概要
  - 車両の自動運転を制御するシステム
- フローチャート

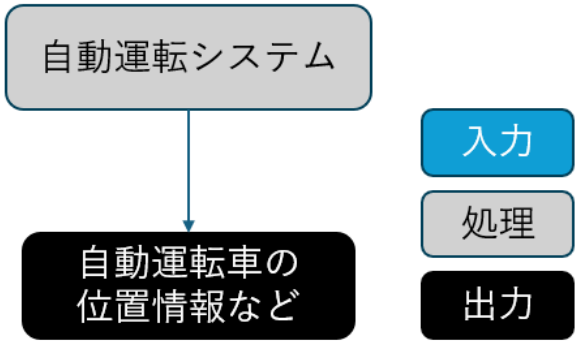


図 3-21 自動運転システムのフローチャート

- データ仕様
  - 入力
    - 自動運転のためのセンサ入力などがあるが、今回の実証実験のスコープ外とする
  - 出力
    - 自動運転車の位置情報
      - 内容
        - 自動運転車が認識している位置・回転の数値データ
        - 座標系は平面直角座標 8 系
      - 形式
        - 位置・回転ベクトルの数値データを CSV で表した文字列を UDP 通信で受信する
      - データ詳細
        - 内部連携インターフェース【IF207】を参照
- 機能詳細
  - 自動運転機能
    - 処理内容
      - ADENU による自動運転処理
      - 自動運転システムの自己位置・姿勢などを UDP 通信で出力する
    - 利用するライブラリ
      - ADENU（ソフトウェア・ライブラリ【SL002】を参照）
    - 利用するアルゴリズム
      - ADENU

13. 【FN013】分析プログラム

- 機能概要
  - 走行後にログに記録された情報を分析する機能
- フローチャート

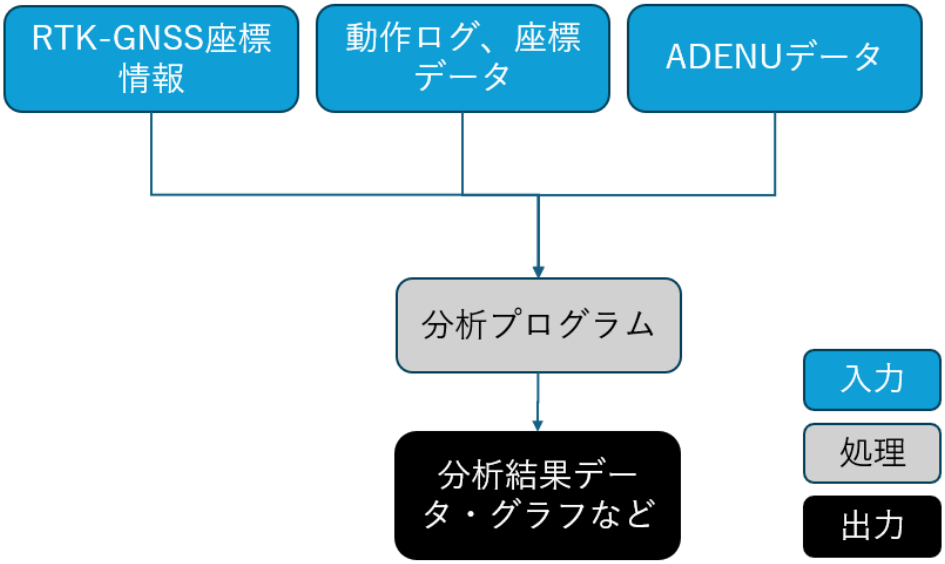


図 3-22 分析プログラムのフローチャート

- データ仕様
  - 入力
    - 動作ログ、座標データ
      - 内容
        - 推定されたカメラ姿勢の位置・回転の数値データ、時刻、計算過程の各種数値など
        - 座標系は地理座標と相互変換可能な内部座標系
      - 形式
        - CSV 形式
      - データ詳細
        - ファイル入力インターフェース【IF007】を参照
    - 自動運転車の位置情報
      - 内容
        - 自動運転車が認識している位置・回転の数値データ
        - 座標系は平面直角座標 8 系
      - 形式
        - CSV 形式
      - データ詳細
        - ファイル入力インターフェース【IF006】を参照
    - RTK-GNSS 座標情報
      - 内容
        - 観測した地理座標、時刻、精度など

- 座標系は WGS84
- 形式
  - CSV 形式
- データ詳細
  - ファイル入力インターフェース【IF005】を参照

- 出力
  - 分析結果の数値やグラフなど
    - 内容
      - 分析結果
    - 形式
      - 数値データ、グラフなど
    - データ詳細
      - 結果については自己位置推定機能の検証 4-2 自己位置推定機能の検証で記載

- 機能詳細
  - 分析プログラム
    - 処理内容
      - ログから精度などを計算する
      - 統計値の算出、グラフ化などを行う
    - 利用するライブラリ
      - Python による自作プログラム群
    - 利用するアルゴリズム
      - なし

14. 【FN014】VPS マップ作成

- 機能概要
  - KdVisual にレンダリング画像を入力することで VPS マップを作成する
- フローチャート

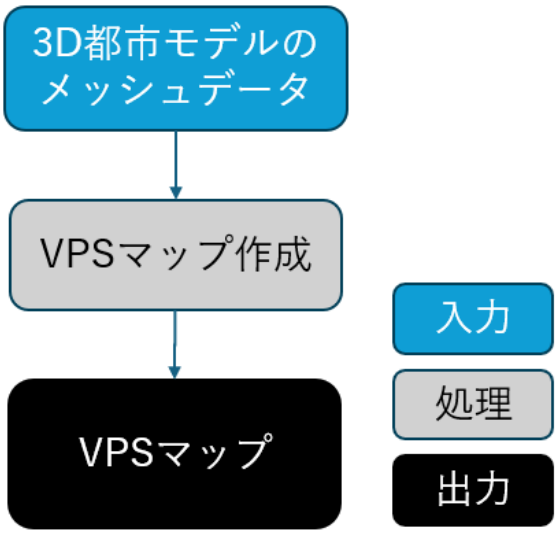


図 3-23 VPS マップ作成機能のフローチャート

- データ仕様
  - 入力
    - 3D 都市モデルのテクスチャ付き 3D メッシュデータ
      - 内容
        - 3D 都市モデルのメッシュデータ
      - 形式
        - 【FN011】により CityGML から OBJ 形式に変換したファイル
      - データ詳細
        - ファイル入力インターフェース【IF001】を参照
  - 出力
    - VPS マップ
      - 内容
        - KdVisual の kdvm 形式マップデータ
      - 形式
        - 独自データ形式
      - データ詳細
        - ファイル出力インターフェース【IF104】を参照
- 機能詳細
  - VisualSLAM
    - 処理内容
      - 入力されたカメラ画像を元に、自己位置の推定と周囲のマップの作成を行う

- 入力された画像の特徴点とその特徴量を計算し、自己位置推定や周辺環境の三次元再構築を行う

✧ 利用するライブラリ
- KdVisual（ソフトウェア・ライブラリ【SL003】を参照）
- ROS（ソフトウェア・ライブラリ【SL004】を参照）

✧ 利用するアルゴリズム
- VisualSLAM（アルゴリズム【AL004】を参照）

➢ VPS マップ作成機能

✧ 処理内容
- Unity 上でカメラ位置を順次移動させながら 3D 都市モデルのレンダリング画像を ROS# 経由で KdVisual に入力する
- KdVisual の VisualSLAM 機能で VPS マップが作成される

✧ 利用するライブラリ
- KdVisual（ソフトウェア・ライブラリ【SL003】を参照）
- ROS（ソフトウェア・ライブラリ【SL004】を参照）
- Unity（ソフトウェア・ライブラリ【SL005】を参照）
- ROS#（ソフトウェア・ライブラリ【SL006】を参照）

✧ 利用するアルゴリズム
- VisualSLAM（アルゴリズム【AL004】を参照）

3-3. アルゴリズム

3-3-1. 利用したアルゴリズム

表 3-4 利用したアルゴリズム一覧

ID	アルゴリズムを利用した機能	名称	説明	選定理由
AL001	FN001	C*	● 実写画像とレンダリング画像同士の NID（Normal Information Distance）【AL002】を用いた画像マッチングによる評価値を、BFGS 法（Broyden–Fletcher–Goldfarb–Shanno algorithm）【AL003】により最小値探索を行うことで自己位置を推定するアルゴリズム	● 3D 都市モデルのジオメトリを活用した VPS アルゴリズムとして妥当性が高いため
AL002	FN001	NID	● 【AL001】C*で用いられる画像同士の類似度の指標を計算するアルゴリズム ● 画像同士の輝度のエントロピーを計算することでロバストな類似度指標とす	● C*で利用されているため

			るもの	
AL003	FN001	BFGS 法	● 【AL001】C*で用いられる最適化アルゴリズム ● 本アルゴリズム自体は準ニュートン法に属する汎用の非制限非線形最適化問題に対する反復的解法の一つ ● C*では、NID の勾配を基にカメラ位置及び姿勢を推定するのに用いられる	● C*で利用されているため
AL004	FN002	Visual SLAM	● 画像から周囲のマップを作りながら自己位置を推定するアルゴリズムの総称 ● KdVisual が実行しているアルゴリズム。詳細は製品となるので非公開	● C*のアルゴリズムを補完するため ● 画像の特徴点をグローバルに検索して自己位置推定を行うため、高精度の初期位置推定を必要としない
AL005	FN006	カルマンフィルタ	● 誤差のある観測値を用いて、対象システムの状態を推定・制御するためのアルゴリズム ● 【FN006】で利用しているアルゴリズム。GPS、IMU などの情報を基に、より確からしい位置を計算し【FN001】C*に入力する	● ロボットなどの自己位置推定で一般的に利用されているアルゴリズムのため
AL101	FN003	推定位置連携アルゴリズム	● 【FN001】C*と【FN002】KdVisual により推定された座標とセンサ情報などを組み合わせて、より確からしい推定座標を出力するアルゴリズム	● 新規開発
AL102	FN001	C*の調整	● 【FN001】C*の処理を、走行しているモビリティに合わせてカスタマイズを行う	● 前年度の C*の挙動の中で可能な改善点だったため

1) 【AL001】C*

C*は国立研究開発法人産業技術総合研究所の大石氏を中心として開発されたアルゴリズムで、カメラからの実写画像と 3D モデルをレンダリングした結果の画像との類似度を基に、カメラの 3D モデル空間内の位置を推定する仕組みである。

既存手法のように、あらかじめ特徴点と特徴量ベクトルを計算した VPS 専用のマップを作る必要がなく、3D モデルの形状とテクスチャに基づいた位置推定が可能である。また、LiDAR などのセンサを使わずに単眼カメラのみで位置推定が可能である。

C*の主要構成要素は以下 3 点である。

- 3D モデルのレンダリングと実写画像との比較
- 画像同士の類似度指標に NID（詳細は 2）に記載）を活用
- 最適化アルゴリズムにより自己位置の推定を実施

また、C*は ROS（Robot Operating System）、Linux、CUDA の環境で動作する。C*は、カメラからの入力画像と 3D モデルをレンダリングした画像の類似度を基にカメラ座標を推定するため、既存の VPS の大半が使っている画像特徴点・特徴量を用いた方式と比べて以下の利点がある。

- 3D モデルのみで自己位置推定が可能。
- 点ではなく面で処理するためノイズやオクルージョンに強い。
- 局所的な画像の特徴だけでなく、3D モデルによる幾何学的形状を自己位置推定に生かすことができる。

図 3-24 に、C*動作の概要を示す。

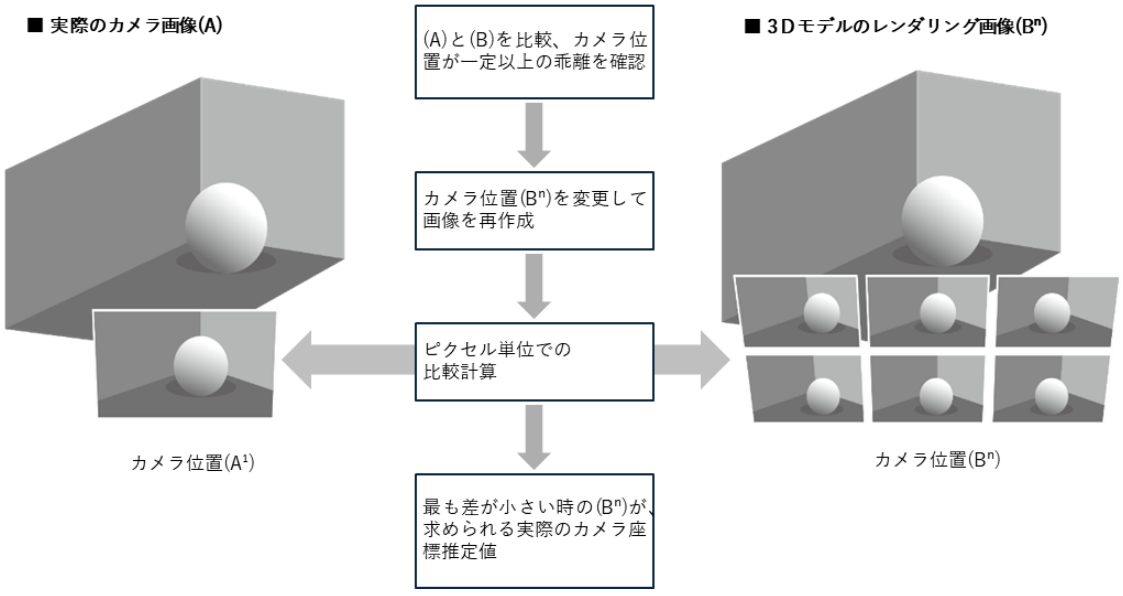


図 3-24 C*の概略動作フロー

今回の実証実験では、図 3-25 のように対照用の 3D モデルとして 3D 都市モデルを用い、都市スケールでの自己位置推定を検証する。

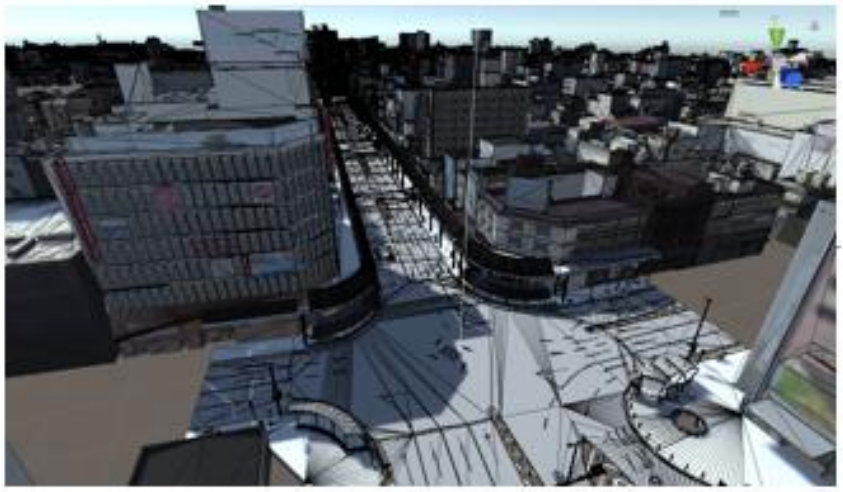


図 3-25 3D 都市モデルをレンダリングしたもの

2) 【AL002】NID

NID は、C*では画像同士の類似度の指標として使われている。下図の手法で計算される。NID は情報量のエントロピーに基づく概念で、ここでは二つの画像の類似度が高いほど低い値になる。直接画素値を比較するものに比べて、照明条件の違いやノイズ、遮蔽などに強く、C*の画像類似度の指標として使われている。

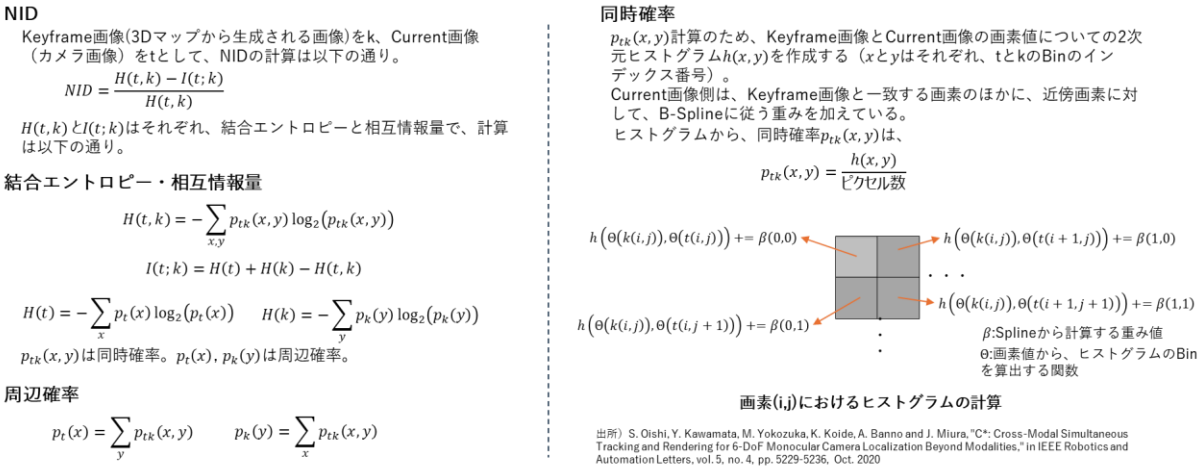


図 3-26 NID 計算式・計算手法

3) 【AL003】BFGS 法

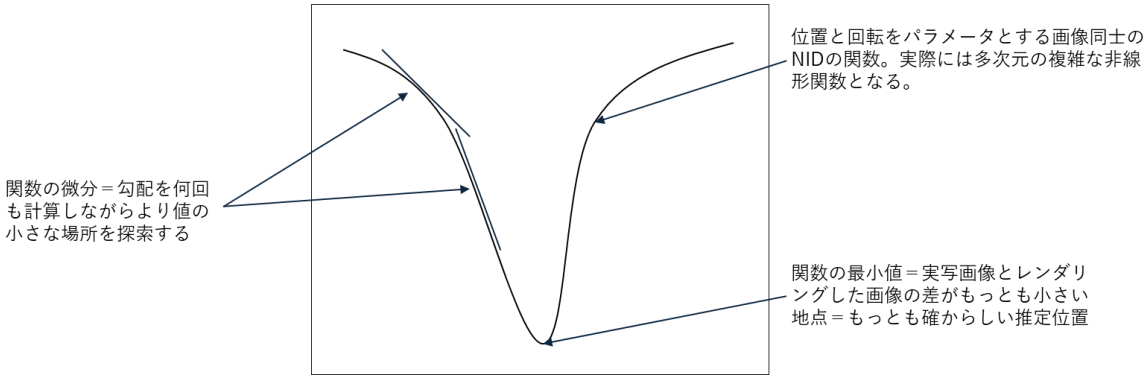


図 3-27 BFGS 法の概念図

C*では最適化アルゴリズムとして BFGS 法を利用して、自己の位置と回転を反復計算により求める。まず、位置と回転を入力するとそのカメラ姿勢でレンダリングした画像と実写画像との NID を出力する関数を作成し、この関数の微分を求め BFGS 法を適用することにより、NID の最小点（カメラ画像とレンダリング画像が最も類似している点）を探索し、解となる位置と回転を推定結果の自己位置とする仕組みとなっている。

4) 【AL004】Visual SLAM（KdVisual）

KdVisual は Kudan 社が開発した SLAM 及び VPS システムである。KdVisual は、カメラ画像を用いた Visual SLAM（Simultaneous Localization and Mapping）で、カメラ画像の特徴点を抽出し、あらかじめ作成しておいた専用の三次元点群マップの特徴点データベースと照合することで、カメラ画像内の点と三次元の点の対応をとり、自己位置の推定を行う仕組みである。詳細なアルゴリズムは非公開となっている。

KdVisual では、最初に SLAM モジュールを用いて、一連の入力画像から KdVisual 専用のマップを作成する。このマップデータを MapOptimiser に入力し処理させることで、マップの最適化を行う。MapOptimiser 内の処理は非公開だが、SLAM で一般的なバンドルアジャストメントなどの処理を行うと想定される。
こうして作成したマップデータは、再度 SLAM モジュールで読み込み、カメラ画像を入力することで、推定された自己位置が出力される VPS 動作を行う。下図に概要を示す。

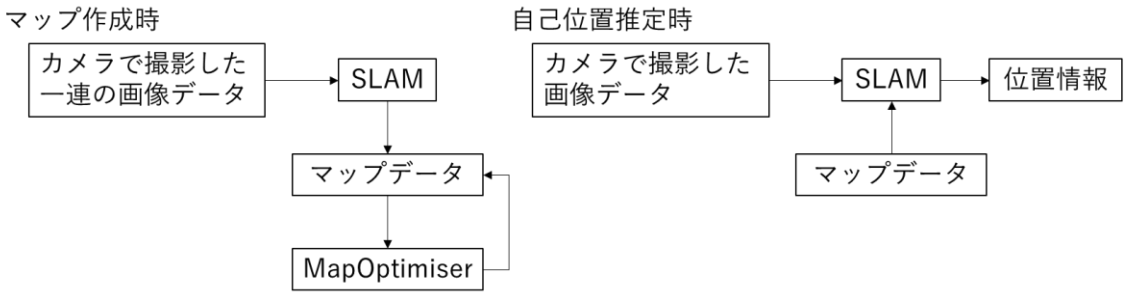


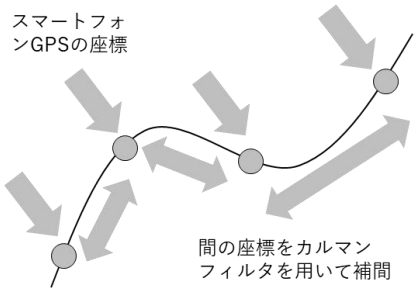
図 3-28 KdVisual の動作の概要

今回の実証実験では、マップデータ作成時に 3D 都市モデルをレンダリングした画像を入力することで、3D 都市モデルを元データとした VPS 動作の検証を行う。

5) 【AL005】カルマンフィルタ

カルマンフィルタは、離散的な誤差のある観測値から時間変化する量を推定するためのアルゴリズムであり、ロケットの軌道推定に始まり、ロボットや自動運転での位置の補間などに活用されている実績のある仕組みである。

カルマンフィルタでは、現在の座標とセンサの観測値を用いて、次の座標を予測し、実際に次の座標が得られたときに予測値との差分を用いて予測パラメータを調整する過程を繰り返す。これにより、精度の高い座標推定と補間が可能になる。



予測

$\hat{\boldsymbol{x}}_{k|k-1} = F_k \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{u}_k$ （今の時刻の予測推定値）

$P_{k|k-1} = F_k P_{k-1|k-1} F_k^{\mathrm{T}} + G_k Q_k G_k^{\mathrm{T}}$ （今の時刻の予測誤差行列）

更新

$\boldsymbol{e}_k = \boldsymbol{z}_k - H_k \hat{\boldsymbol{x}}_{k|k-1}$ （観測残差、innovation）

$S_k = R_k + H_k P_{k|k-1} H_k^{\mathrm{T}}$ （観測残差の共分散）

$K_k = P_{k|k-1} H_k^{\mathrm{T}} S_k^{-1}$ （最適 カルマンゲイン）

$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k \boldsymbol{e}_k$ （更新された状態の推定値）

$P_{k|k} = (\mathrm{I} - K_k H_k) P_{k|k-1}$ （更新された誤差の共分散）

カルマンフィルタの数式
カルマンフィルター - Wikipedia

図 3-29 カルマンフィルタの概要

3-3-2. 開発したアルゴリズム

1) 【AL101】推定位置連携アルゴリズム

- 本アルゴリズムを利用した機能
  - 【FN003】
- 実装の詳細
  - C*と KdVisual の二つの座標値の入力から、より確からしい座標推定を行う。
  - 方針①として、FN006：初期位置入力機能でも用いているカルマンフィルタを用いる。
  - 方針②として、初期位置入力に KdVisual を用い、C*により継続的トラッキングを行う。
  - 上記①、②の実装を検討し、最適なものを選択する。

2) 【AL102】C*の調整

- 本アルゴリズムを利用した機能
  - 【FN001】
- 実装の詳細
  - 継続的なトラッキングを可能とするように、各フレームでの初期位置に対してカルマンフィルタなどでより良い推定を行うことを検討する。
  - 2023 年 5 月に産業技術総合研究所の C*の拡張実装である、「L-C*」が発表されたので、参考にする。

3-4. データインターフェース

3-4-1. ファイル入力インターフェース

1) 【IF001】3D 都市モデルのテクスチャ付きメッシュデータ（結合モデル）

対象地域の 3D 都市モデルの建築物・都市設備・道路の CityGML データを OBJ 形式と PNG 画像に変換したもの。C*の対応形式が OBJ 形式のみであること、今回使用する全てのソフトウェアで OBJ 形式の読み込みに対応していることから、本データを共通の 3D モデルデータとして使用する。

- 本インターフェースを利用した機能
  - 【FN001】
- ファイル形式
  - OBJ 形式、PNG 形式

2) 【IF002】3D 都市モデルのメッシュデータ（地形）

対象地域の 3D 都市モデルの地形の CityGML データを OBJ 形式に変換したもの。GPS 座標から高さを取得する際に地表高の参考値として使用する。

- 本インターフェースを利用した機能
  - 【FN004】
- ファイル形式
  - OBJ 形式

3) 【IF003】VPS マップ

KdVisual が使用するマップデータ。Unity 上で開発したプログラムから、あらかじめ設定した経路に沿って連続的に 3D 都市モデルをレンダリングした画像を KdVisual に入力し、その画像群をもとに KdVisual の VSLAM 機能で特徴点の三次元マップを作成したものを保存したデータ。

- 本インターフェースを利用した機能
  - 【FN002】
- ファイル形式
  - KdVisual 独自バイナリ形式

4) 【IF004】3D 都市モデル CityGML ファイル

3D 都市モデルの CityGML データ。対象地域の建築物 LOD2,3、都市設備 LOD3、道路 LOD3、地形 LOD2 の CityGML を使用した。

- 本インターフェースを利用した機能
  - 【FN011】
- ファイル形式
  - CityGML 形式

5) 【IF005】RTK-GNSS 座標情報

Drogger RWP により取得した高精度位置情報のデータ。ソフトバンク株式会社の ichimill サービスにより基準局情報を取得し、RTK 測位により数 cm の測位誤差で計測した座標情報を記録したもの。

- 本インターフェースを利用した機能
  - 【FN013】
- ファイル形式
  - CSV テキスト形式

6) 【IF006】ADENU データ（ファイル形式）

実証時に本システムを搭載した自動運転車の自動運転システムである ADENU が記録した車両位置や姿勢のデータをファイルに記録したもの。

- 本インターフェースを利用した機能
  - 【FN013】
- ファイル形式
  - CSV テキスト形式

7) 【IF007】各種動作ログ、座標データ等

実行時に本システムの各機能が出力するログ。システムの動作状況や推定した座標、入力されたデータなどを適宜テキストファイルに出力する。本システムはコンポーネント間の接続をネットワーク経由でおこなっているため、接続が切断された際にもデータの喪失がおこらないように、各コンポーネントでローカルファイルにログを記録するようにしている。

- 本インターフェースを利用した機能
  - 【FN013】
- ファイル形式
  - テキスト形式

3-4-2. ファイル出力インターフェース

1) 【IF101】3D 都市モデルのテクスチャ付きメッシュデータ（結合モデル）

対象地域の 3D 都市モデルの建築物・都市設備・道路の CityGML データを OBJ 形式と PNG 画像に変換したもの。C*の対応形式が OBJ 形式のみであること、今回使用する全てのソフトウェアで OBJ 形式の読み込みに対応していることから、本データを共通の 3D モデルデータとして使用する。

- 本インターフェースを利用した機能
  - 【FN001】【FN008】【FN009】【FN014】
- ファイル形式
  - OBJ 形式、PNG 形式

2) 【IF102】3D 都市モデルのメッシュデータ（地形）

対象地域の 3D 都市モデルの地形の CityGML データを OBJ 形式に変換したもの。GPS 座標から高さを取得する際に地表高の参考値として使用する。

- 本インターフェースを利用した機能
  - 【FN004】
- ファイル形式
  - OBJ 形式

3) 【IF103】ADENU データ（ファイル記録）

実証時に本システムを搭載した自動運転車の自動運転システムである ADENU が記録した車両位置や姿勢のデータをファイルに記録したもの。

- 本インターフェースを利用した機能
  - 【FN013】
- ファイル形式
  - CSV テキスト形式

4) 【IF104】VPS マップ

KdVisual が使用するマップデータ。Unity 上で開発したプログラムから、あらかじめ設定した経路に沿って連続的に 3D 都市モデルをレンダリングした画像を KdVisual に入力し、その画像群をもとに KdVisual の VSLAM 機能で特徴点の三次元マップを作成したものを保存したデータ。

- 本インターフェースを利用した機能
  - 【FN002】
- ファイル形式
  - KdVisual 独自バイナリ形式

5) 【IF105】各種アプリケーション動作ログ、座標データ等

実行時に本システムの各機能が出力するログ。システムの動作状況や推定した座標、入力されたデータなどを適宜テキストファイルに出力する。本システムはコンポーネント間の接続をネットワーク経由でおこなっているため、接続が切断された際にもデータの喪失がおこらないように、各コンポーネントでローカルファイルにログを記録するようにしている。

- 本インターフェースを利用した機能
  - 【FN013】
- ファイル形式
  - テキスト形式

3-4-3. 内部連携インターフェース

1) 【IF201】スマートフォンカメラからの画像データ

スマートフォンのカメラで撮影した画像を初期位置入力プログラムで ROS メッセージに変換したもの。ネットワーク帯域の節約のため JPEG 形式で圧縮している。

- 本インターフェースを利用した機能
  - 【FN001】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - CompressedImage
- トピック
  - /mynteye/left/image_mono/compressed
  - /mynteye/right/image_mono/compressed

2) 【IF202】初期位置入力の画像及び座標データ

初期位置入力プログラムで取得した GPS 座標から座標変換等の処理をしてカメラ画像と合わせて ROS メッセージに変換したもの。姿勢付きの画像データの形式で送信される。

- 本インターフェースを利用した機能
  - 【FN001】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - PosedImage
- トピック
  - /mynteyed_localizer/pose_refinement

3) 【IF203】推定された位置情報

C*により推定された自己位置の座標情報。位置と姿勢を格納した ROS メッセージ。

- 本インターフェースを利用した機能
  - 【FN001】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - TF
- トピック
  - /tf

4) 【IF204】カメラからの画像データ

PC に接続したカメラからの画像データ。RealSense D455、TIAR IV C1 を選択し接続する。物理接続は USB。ROS の機能によりカメラ画像が ROS メッセージとして配信される。

- 本インターフェースを利用した機能
  - 【FN002】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - CompressedImage
- トピック
  - /camera/infra1/image_rect_raw/compressed
  - /camera/infra2/image_rect_raw/compressed

5) 【IF205】推定された位置情報

KdVisual により推定された自己位置の座標情報。位置と姿勢を格納した ROS メッセージ。

- 本インターフェースを利用した機能
  - 【FN002】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - TF
- トピック
  - /kdvisual_ros/pose

6) 【IF206】推定された位置情報

C*と KdVisual の推定した座標情報を推定位置連携機能により連携した座標。位置と姿勢を格納した ROS メッセージ。

- 本インターフェースを利用した機能
  - 【FN003】
- プロトコル
  - ROS メッセージ
- メッセージタイプ
  - TF
- トピック
  - /tf_processed

7) 【IF207】自動運転車の位置情報

自動運転システム ADENU が認識している自動運転車の位置・姿勢情報をリアルタイムに UDP 通信で受信す

る。データはパケット内に CSV 文字列としてエンコードされている。

- 本インターフェースを利用した機能
  - 【FN009】
- プロトコル
  - 独自の UDP を使用したプロトコル
- メッセージタイプ
  - UDP により、データを CSV 文字列にしたものを受信する

8) 【IF208】スマートフォンのカメラ画像

スマートフォンのカメラ画像を Unity のメソッドで取得したもの。ピクセルの値を格納したバイナリ配列で出力される。

- 本インターフェースを利用した機能
  - 【FN005】【FN006】【FN007】
- プロトコル
  - UnityAPI 呼び出し
- メッセージタイプ
  - ピクセル値を格納したバイナリ配列

9) 【IF209】スマートフォンの GPS 情報

スマートフォンに搭載された GPS 機能により取得した現在地の座標。精度は数 m～数十 m。Unity のメソッドで取得したもの。

- 本インターフェースを利用した機能
  - 【FN004】
- プロトコル
  - ネイティブ API 呼び出し
- メッセージタイプ
  - 緯度経度と高さの 3 つの Double 変数

10) 【IF210】スマートフォンの IMU データ

スマートフォンに搭載された IMU センサーにより取得した、加速度、ジャイロ、コンパスの情報。それぞれ Vector3 型として Unity のメソッドで取得する。

- 本インターフェースを利用した機能
  - 【FN006】
- プロトコル
  - UnityAPI 呼び出し
- メッセージタイプ
  - 加速度、角速度、コンパスのそれぞれの Vector3 変数

11）【IF211】座標変換結果の座標・姿勢

座標変換機能により変換された座標をメソッド呼び出しで受け渡す。

- 本インターフェースを利用した機能
  - 【FN005】
- プロトコル
  - メソッド呼び出し
- メッセージタイプ
  - 座標・姿勢を格納した Vector3、Quaternion の変数

12）【IF212】NID 探索結果の座標・姿勢

NID 探索機能により変換された座標をメソッド呼び出しで受け渡す。

- 本インターフェースを利用した機能
  - 【FN006】
- プロトコル
  - メソッド呼び出し
- メッセージタイプ
  - 座標・姿勢を格納した Vector3、Quaternion の変数

3-4-4. 外部連携インターフェース

本プロジェクトにおいて、外部とのデータ連携は実施しない。

3-5. 実証に用いたデータ

3-5-1. 活用したデータ一覧

1） 利用した 3D 都市モデル

- 年度：2021 年度
- 都市名：沼津市
- ファイル名：22203_numazu-shi_2021_citygml_4_op.zip
- メッシュ番号：5238-5628, 5238-5629, 5238-5618, 5238-5619, 5238-5608, 5238-5609, 5238-4698, 5238-4699

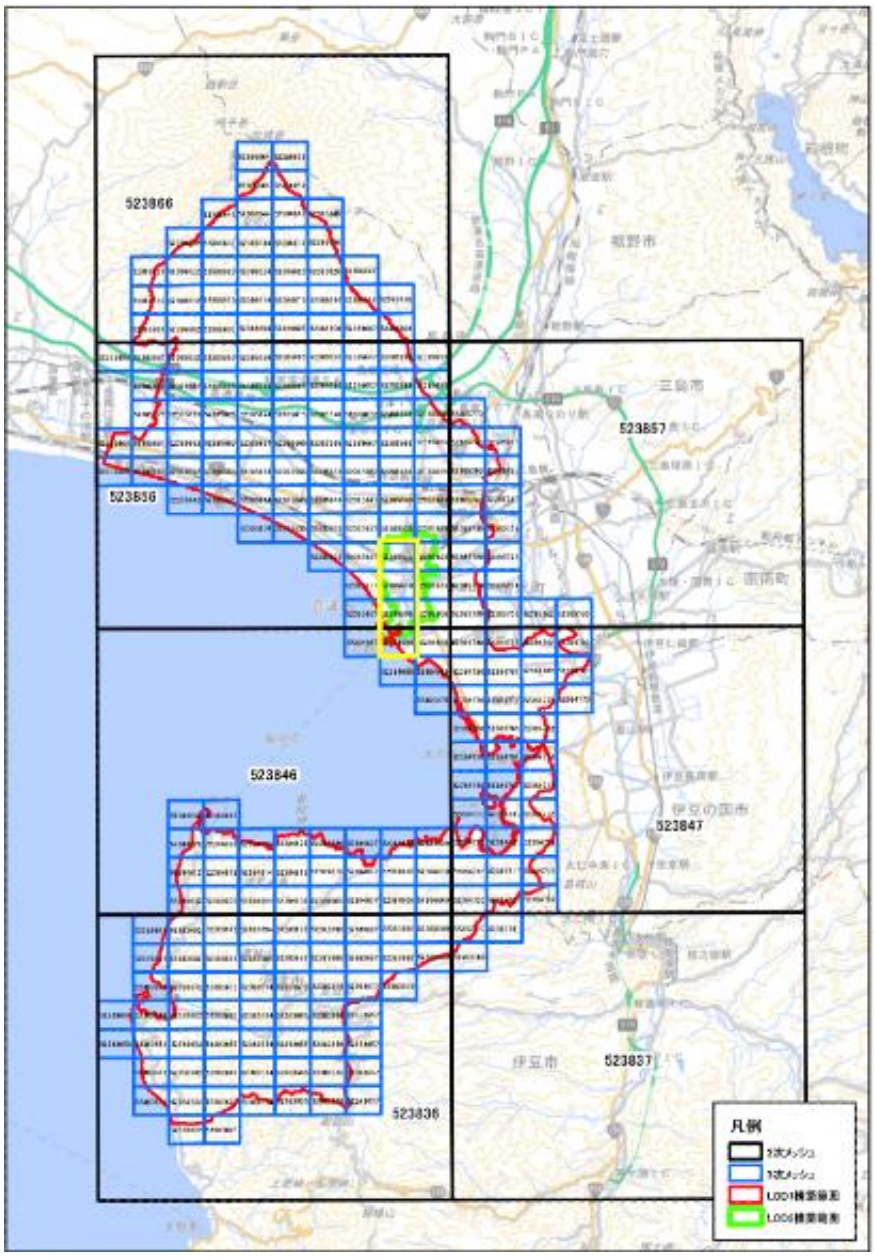


図 3-30 インデックスマップ – 全体像（沼津市）

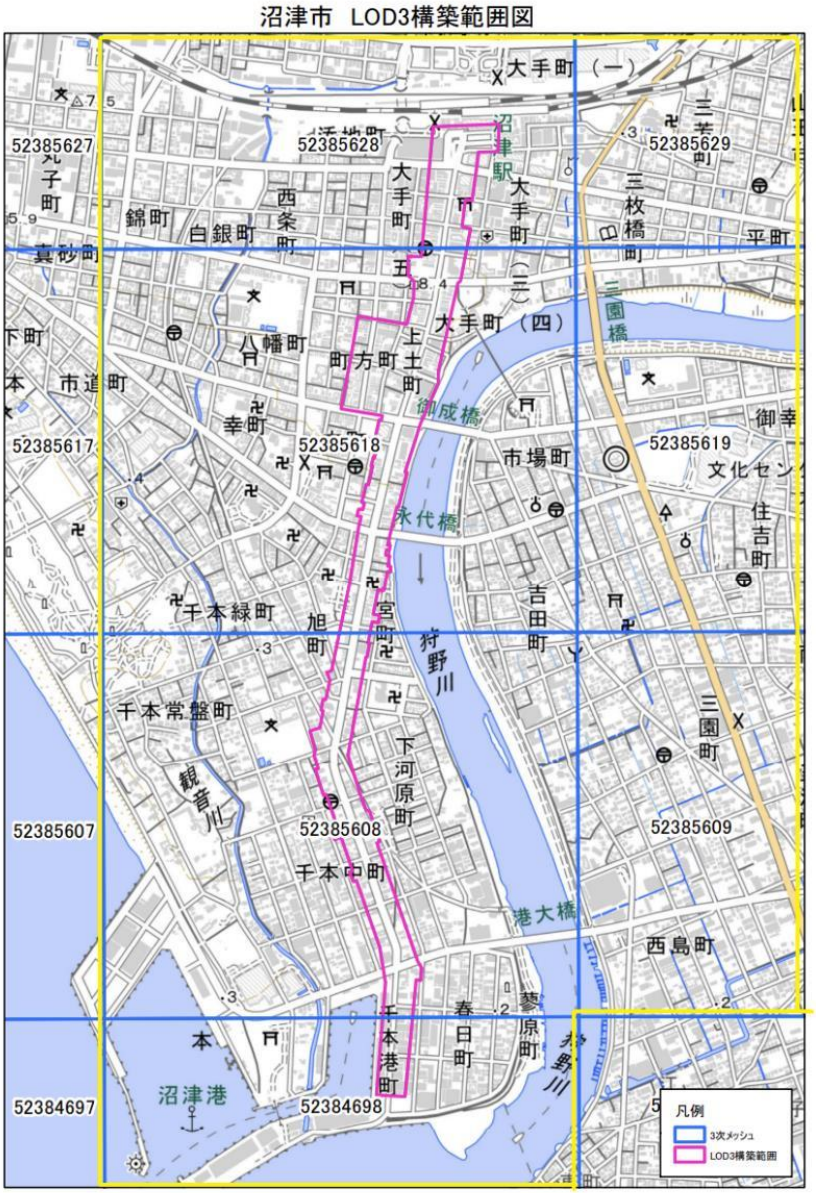


図 3-31 インデックスマップ – 詳細（沼津市）

表 3-7 利用した 3D 都市モデル

地物	地物型	属性区分	属性名	利用想定	データを利用した機能（ID）
建築物 LOD2	bldg:Building	空間属性	bldg:lod2Solid	建築物の LOD2 の立体、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
	app:Appearance	空間属性	app:imageURI など	建築物の LOD2 テクスチャ、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
建築物 LOD3	bldg:Building	空間属性	bldg:lod3Solid	建築物の LOD3 の立体、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
	app:Appearance	空間属性	app:imageURI など	建築物の LOD3 テクスチャ、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
道路 LOD3	tran:Road	空間属性	tran:lod3MultiSruface	道路の LOD3 立体、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
都市設備 LOD3	frn:CityFurniture	空間属性	frn:lod3Geometry	都市設備の LOD3 立体、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
	app:Appearance	空間属性	app:imageURI など	都市設備の LOD3 テクスチャ、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010
地形 LOD2	dem:TINRelief	空間属性	dem:tin	地形の LOD2 立体、VPS 用マップとして利用	FN001、FN002、FN005、FN008、FN010

2) 利用したその他のデータ

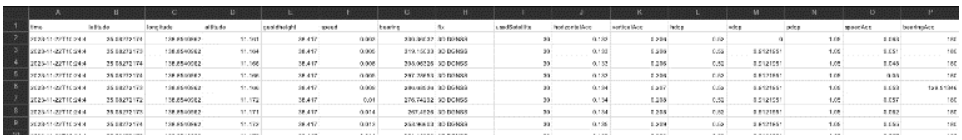
1. データ一覧

表 3-8 利用したその他データ（一覧）

ID	エリア（都市）	活用データ	内容	データ形式	更新情報	出所	データを利用した機能（ID）
DT101	-	スマートフォン GPS	スマートフォンの GPS で取得した現在地座標	地理座標の数値	-	端末	FN004
DT102	-	スマートフォン IMU	スマートフォンのセンサで取得した加速度と角速度	数値	-	端末	FN006
DT103	-	RTK-GNSS	RTK-GNSS で取得した現在地座標	CSV ファイル	-	RTK-GNSS 受信機	FN013
DT104	-	ADENU データ	ADENU で認識している自己位置の座標など（表示・データ分析用）	CSV、ROSBAG	-	ADENU	FN013
DT105	-	ADENU IMU データ	ADENU で取得した加速度と角速度	CSV 形式データを UDP で受信	-	ADENU	FN006 、FN010

2. データサンプル（イメージ）

表 3-9 利用したその他データ（サンプル）

ID	活用データ	サンプル・イメージ
DT101	スマートフォン GPS	Double lat,lon;
DT102	スマートフォン IMU	Vector3 acc,gyro;
DT103	RTK-GNSS	
DT104	ADENU データ	以下の項目が格納された CSV、及び ROS 動作を記録した bag ファイル position time(sec) position time(nsec) global position x global position y global position z global pose w global pose x global pose y global pose z lane index lane sub index lane pose x lane pose y longitude latitude car state time(sec) car state time(nsec) cotrol mode speed(m/s) heading value type heading value(rad) blinker type gear type battery level battery volts battery currents

		command time(sec) command time(nsec) command mode speed(m/s) heading value type heading value(rad) blinker type gear type horn flag remote time(sec) remote time(nsec) remote mode speed(m/s) steer angle(deg) option_flags imu time(sec) imu time(nsec) acceleration x(m/s^2) acceleration y(m/s^2) acceleration z(m/s^2) angular velocity x(rad/s) angular velocity y(rad/s) angular velocity z(rad/s)
DT105	ADENU IMU データ	Time(sec),Time(nsec),X,Y,Z,Lat,Lon,Speed(m/s),Pose(w),Pose(x),Pose(y),Pose(z),acc_x(m/s2)acc_x(m/s2),acc_y(m/s2),acc_z(m/s2),roll(rad/s),pitch(rad/s),yaw(rad/s) 1671591521,615161356,-99524.011,32777.805,9.203,35.1023662,138.8595421,0,0.88023,0,0,-0.47454,1.15651,-0.09351,9.73925,0.00066,0.00164,0.0017 1671591521,715155609,-99524.011,32777.805,9.203,35.1023662,138.8595421,0,0.88023,0,0,-0.47454,1.1631,-0.09721,9.712,0.00501,-0.00391,-0.00249 1671591521,815136096,-99524.011,32777.805,9.203,35.1023662,138.8595421,0,0.88023,0,0,-0.47454,1.15462,-0.09643,9.729,0.00207,0.00359,0.00271

3-5-2. 生成・変換したデータ

表 3-10 生成・変換したデータ

ID	システムに入力するデータ（データ形式）	用途	処理内容	データ処理ソフトウェア	活用データ（データ形式）	データを利用した機能（ID）
DT201	3D 都市モデル（OBJ 形式＋PNG 形式）	VPS マップの構築、データ表示等のため	● 3D 都市モデル（CityGML）から、PLATEAU SDK for Unity を利用して、OBJ＋PNG の形式に変換、Unity で利用 ● 座標系は、平面直角座標 8 系で実証エリア内に設定した原点へのオフセット計算を行ったもの	独自プログラム	3D 都市モデル（CityGML 形式）	FN001、FN002、FN005、FN008、FN010
DT202	推定座標ログ（テキスト形式）	結果分析	● 各機能の出力座標などをファイルに書き出したもの ● 実証終了後にこれらの情報を元に精度などを評価する。	今回の実証実験の各機能を実装したソフトウェアが出力。Python で読み込み分析結果を出力する。	CSV テキストファイル形式	FN001、FN002、FN003、FN004、FN005、FN006、FN013
DT203	各プログラムのログなど（テキスト形式）	動作確認	● 各機能を実装したソフトウェアが出力するログ	今回の実証実験の各機能を実装したソフトウェアが出力。動作確認などに利用する。	CSV テキストファイル形式	FN001、FN002、FN003、FN004、FN005、FN006、FN013
DT204	KdVisual 用 VPS マップ（独自バイナリ形	VPS による自己位置推定	● 3D 都市モデルをレンダリングした画像を	KdVisual	3D 都市モデル（OBJ 形式＋PNG 形式）	FN002

	式）		KdVisual の VSLAM モジュールに入力しマッピングされた三次元特徴点のデータを保存したもの			

【DT204】KdVisual 用 VPS マップの作成方法

本来は KdVisual に実写のカメラ映像を入力して行う VSLAM を、代わりにレンダリング画像を入力することで 3D 都市モデルをもとにした VPS マップを作成する。
Unity 上に【DT201】3D 都市モデル（OBJ 形式＋PNG 形式）を読み込み、あらかじめ対象ルートの走行を模したカメラ移動用アニメーションパスを作成する。ROS#により KdVisual と接続しカメラをパスに沿って移動させながらレンダリングした画像を随時 KdVisual に送信する。

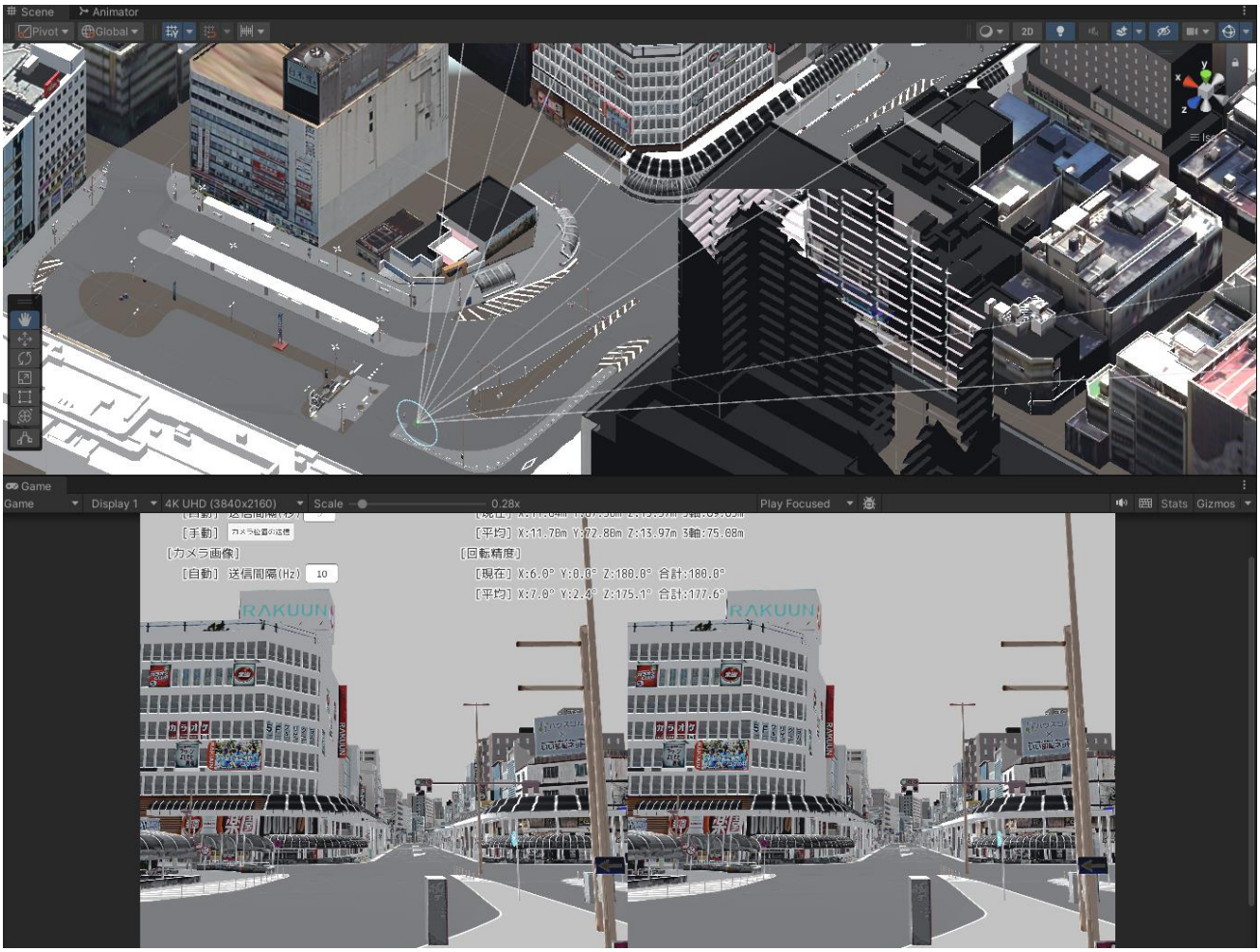


図 3-32 3D 都市モデルのレンダリング画像を KdVisual に送信

この画像を受信した KdVisual は VSLAM モジュールにより画像から三次元の特徴点マップを作成する。車線などを変えて何回か走行し十分マップができたら、これをファイルに保存し、VPS マップとする。

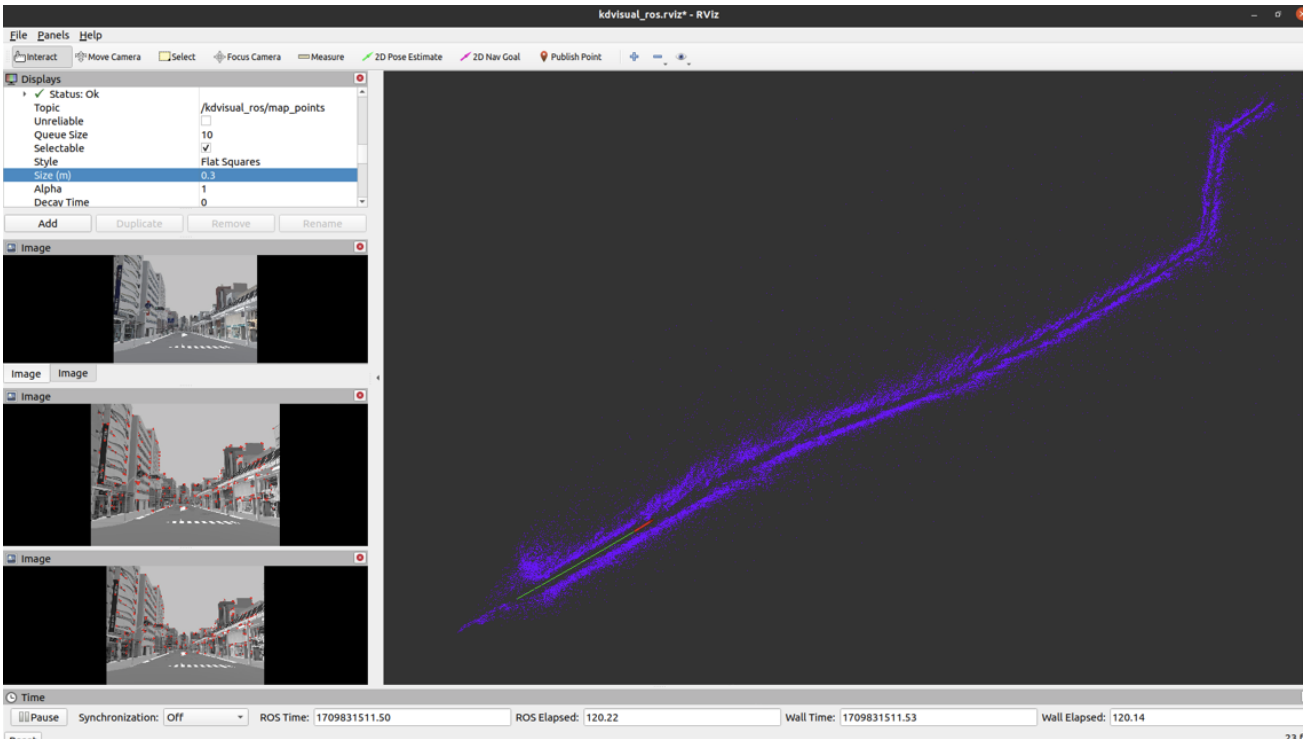


図 3-33 KdVisual 側、入力されたレンダリング画像から VSLAM を行いマップが作成されている

3-6. ユーザーインターフェース

3-6-1. 画面一覧

1） PC 用画面

表 3-11 PC 画面一覧

ID	連携（ID）	画面名	画面説明	画面を表示した機能（ID）
SC001	-	Status Display	● C*、KdVisual の自己位置推定と自動運転車両の位置を、3D 都市モデル上に表示する	FN008、FN009、FN010

※本システムは、機能を変更する際に GUI ではなく CLI を使うので、ここでは Status Display のみについて記載する。

3-6-2. 画面遷移図

画面遷移なし

3-6-3. 各画面仕様詳細

1） PC 用画面

1. 【SC001】Status Display

- 画面の目的・概要
  - C*と KdVisual の自己位置推定結果を 3D 都市モデル上に表示する画面。
  - 3D ビューと各種ステータス表示や状況表示を行う。
- 画面イメージ

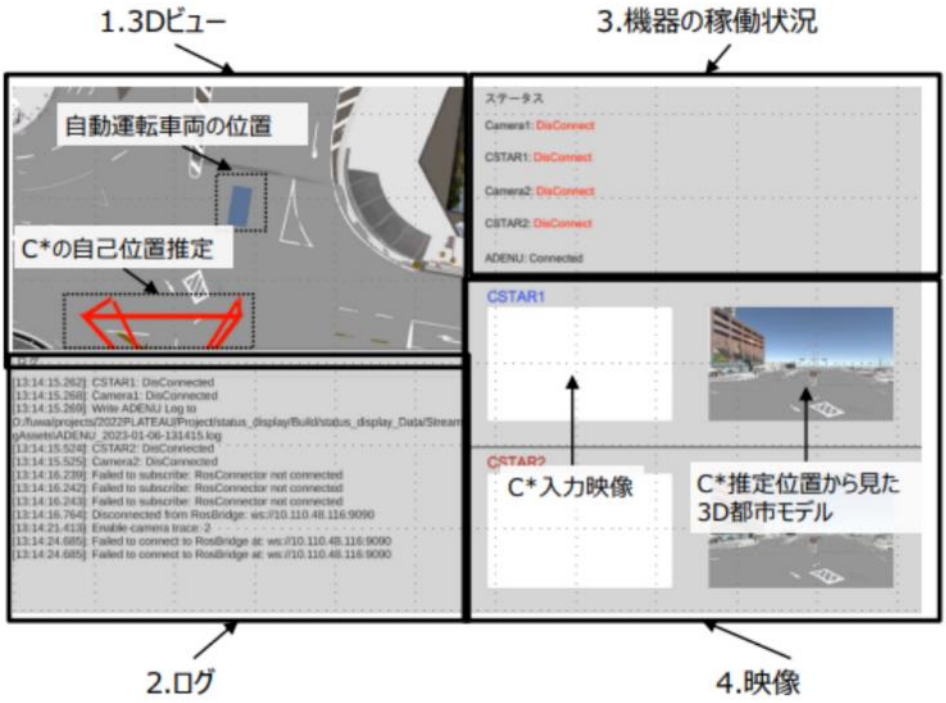


図 3-34 Status Display のイメージ

表 3-12 Status Display 画面内項目

ID	項目	詳細
1	3D ビュー	C*と Kdvisual の自己位置推定結果と自動運転車両の位置を 3D 都市モデル上に表示する
2	ログ	Status Display アプリのログを表示する
3	機器の稼働状況	スマートフォン、位置処理推定 PC、ADENU の稼働状況を表示する
4	映像	C*に入力された映像と、C*・KdVisual の推定位置視点での 3D 都市モデルを表示する

3-7. 実証システムの利用手順

3-7-1. 実証システムの利用フロー

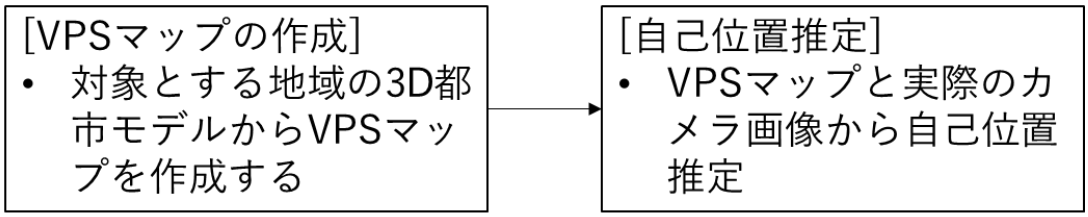


図 3-35 システムの利用フロー

- 3D 都市モデルから VPS マップを作成し、実際のカメラ画像から自己位置推定する。

3-7-2. 各画面操作方法

1) VPS マップの作成

LinuxPC（図では仮想マシンで動作）でコマンドライン操作を用いて KdVisual を作動させ、Unity 上で連続的に移動しながら 3D 都市モデルをレンダリングした画像を送出することで、VPS マップを作成する。

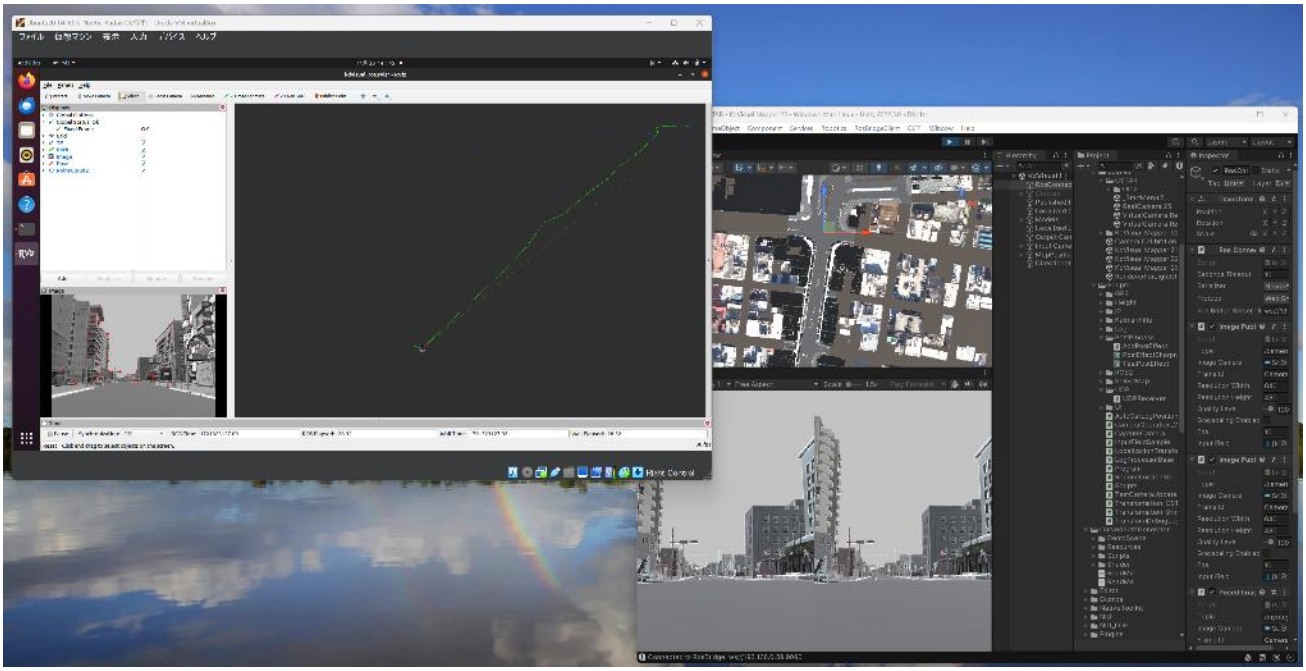


図 3-36 KdVisual での VPS マップ作成

C*は、3D モデルをそのまま VPS マップとして利用するためマップ作成工程はない。

2) 自己位置推定

C*、KdVisual ともに、LinuxPC 上でコマンドライン操作を用いて起動する。その際に起動パラメータとしてマップファイルを指定する。

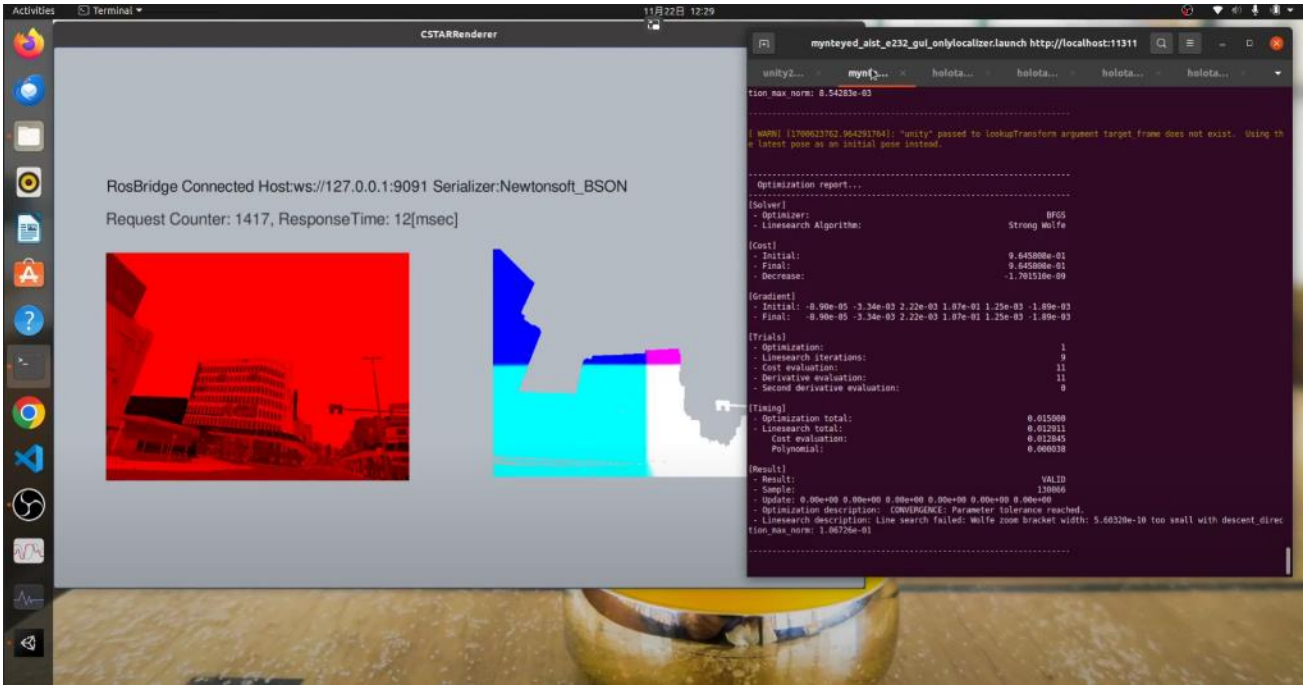


図 3-37 C*の自己位置推定画面

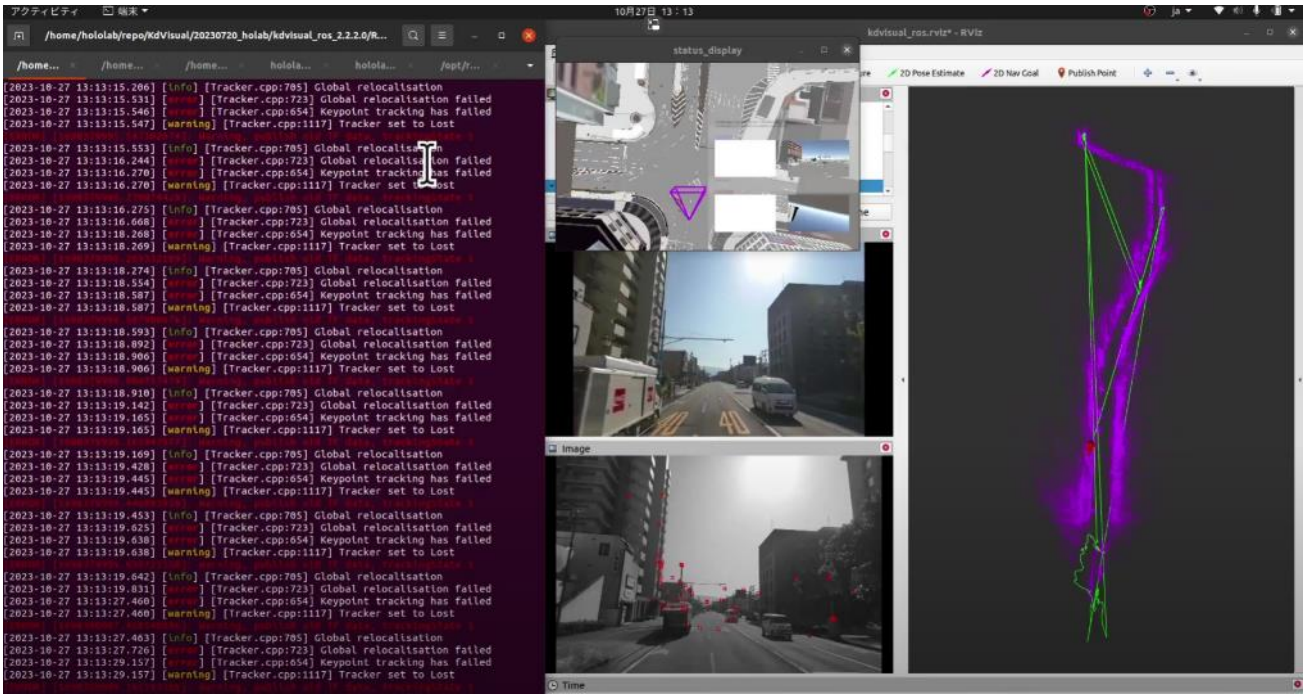


図 3-38 KdVisual の自己位置推定画面と Status Display 動作画面

4. 車両向け自律走行システム：実証技術の検証

4-1. KdVisual による VPS アルゴリズムの検証

4-1-1. 検証目的

- 今年度の検証で新しく導入する KdVisual がゲームエンジンからレンダリングした画像で動作することを確認するために、KdVisual に 3D 都市モデルをレンダリングした映像を入力し、SLAM モジュールよる VPS マップを作成する。
- SLAM による三次元再構成の結果の点群が、3D 都市モデルのジオメトリと見た目上で大きな乖離がないことを確認する。

4-1-2. KPI

表 4-1 KPI 一覧

No.	評価指標・KPI	目標値	目標値の設定理由	検証方法
1	3D 都市モデルのジオメトリと比較	（数値目標はなし）	精度検証で詳細な位置精度を検証するため、ここでは目視による形状や位置の違いを確認する	二つのデータを表示し確認

4-1-3. 検証方法と検証ルート

- 検証方法

1） 3D 都市モデルのジオメトリと比較

事前に Unity でレンダリングした 3D 都市モデルの画像を KdVisual の SLAM モジュールに入力して VPS 用マップを作成する。作成した VPS 用マップの点群を Rviz（ROS の可視化ツール）で表示し、3D 都市モデルのジオメトリと目視で比較して、位置などに大きな違いがないことを確認する。 VPS 用マップは、特徴点の座標と特徴量ベクトルのデータを記録した特殊なバイナリ形式であり、3D 都市モデルとの直接的な比較ができないため、本方法で確認する。
図 4-1 のように、Unity 上で 3D 都市モデルをレンダリングした画像を、Ros-Sharp 経由で、別マシン上で動作している KdVisual に入力し、図 4-2 で表示されているように作成された VPS マップを表示し確認する。

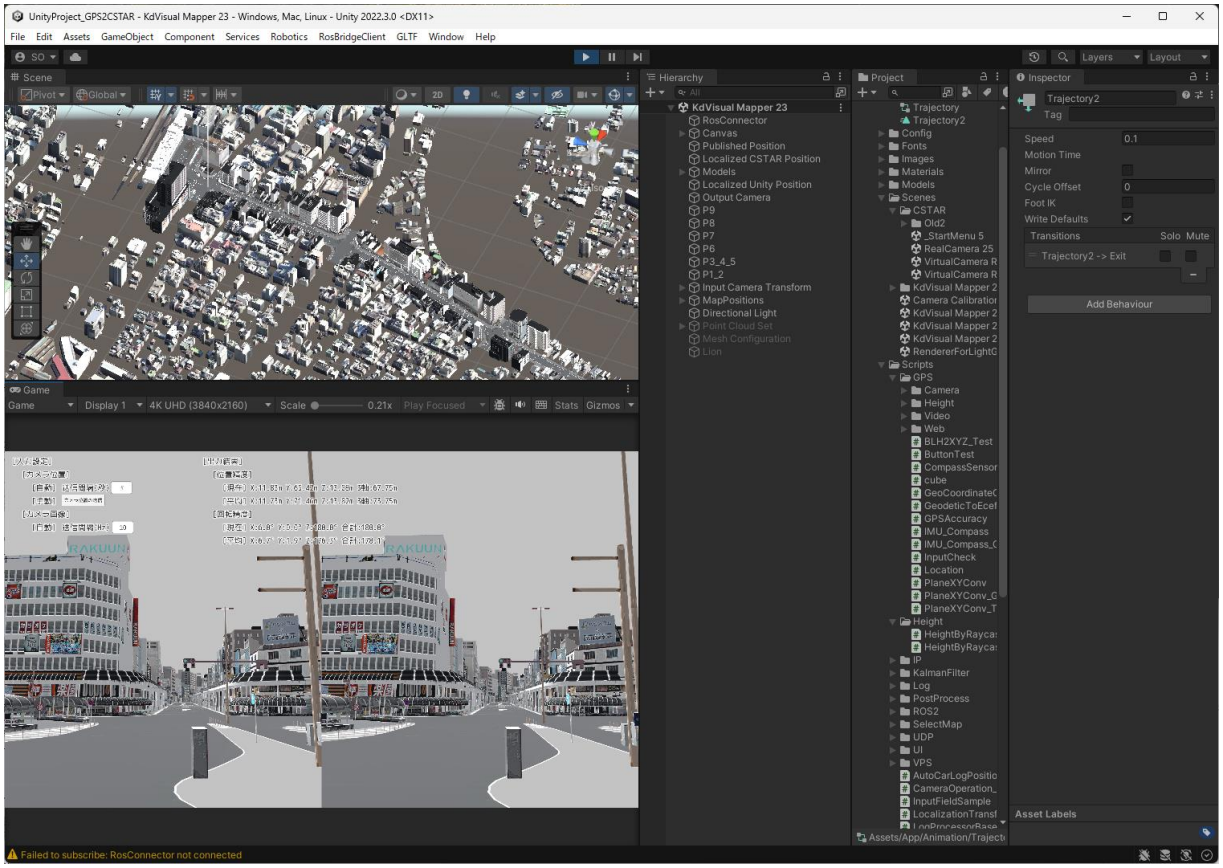


図 4-1 Unity 上で 3D 都市モデルをレンダリング

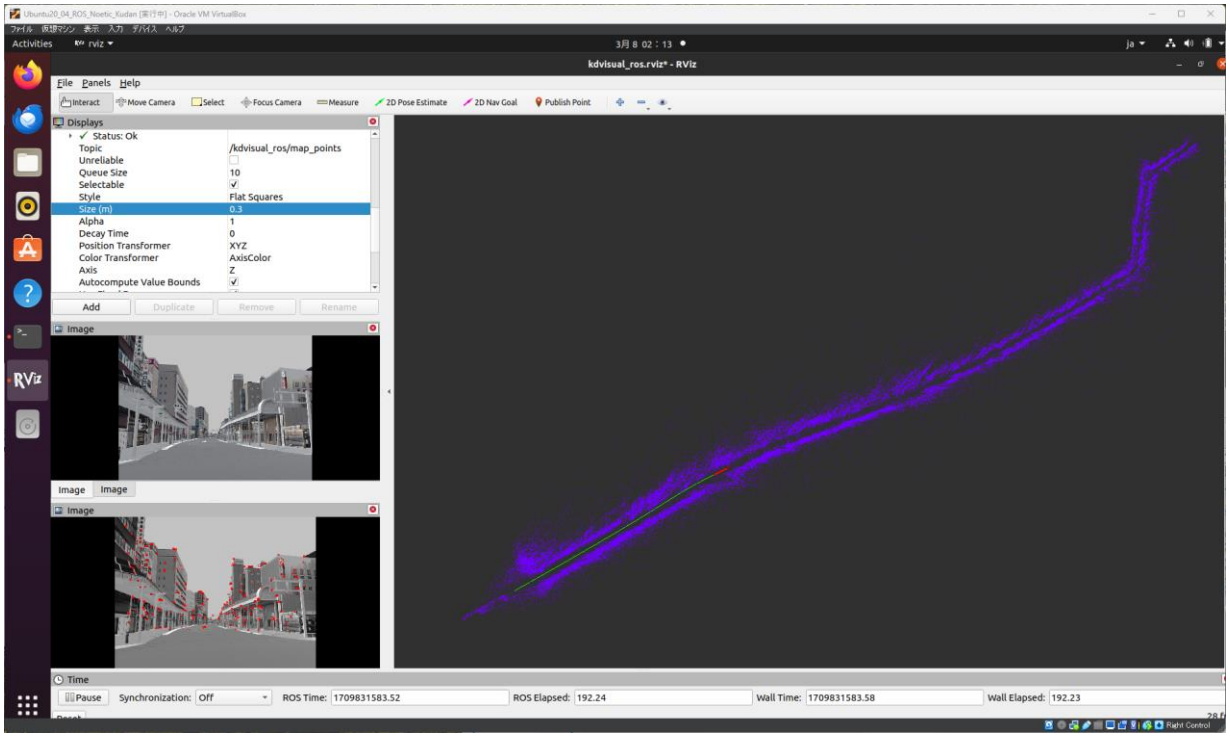


図 4-2 Rviz で表示した VPS マップの点群

- 検証ルート

実際に KdVisual をマップ作成のための SLAM モードで動作させた PC に対して、Unity のプログラムから 3D 都市モデルをレンダリングした映像を入力しマップ作成の後、マップデータを 3D 表示し確認する。
これを沼津駅前→沼津港と沼津港→沼津駅前の 2 方向で実際に走行し、対象ルートに沿って Unity 上でもカメラを動かしながら映像を取得して比較する。

表 4-2 検証ルート一覧

No.	使用した VPS マップ	ルート
1	unity20231122-001.kdvm	沼津駅前→沼津港
2	unity20231122-002.kdvm	沼津港→沼津駅前

4-1-4. 検証結果

1) 3D 都市モデルのジオメトリと比較

- 検証結果まとめ
  - 3D 都市モデルを Unity でレンダリングした画像を入力し KdVisual の SLAM モジュールで作成した VPS 用マップは、十分に現実を反映したものとなり KPI を達成した。

表 4-3 検証結果サマリー

黄セル：KPI 達成	青セル：KPI 未達

検証内容	評価指標・KPI	目標値	結果		示唆
			項目	評価	
3D 都市モデルから作成したマップデータの確認	3D 都市モデルのジオメトリと比較し大きな違いがない	（目視確認による違いの確認）	検証ルート 1	達成	● Unity 上の 3D 都市モデルをレンダリングした画像による SLAM は十分に正確に動作し、マップ作成が達成できた。
			検証ルート 2	達成	

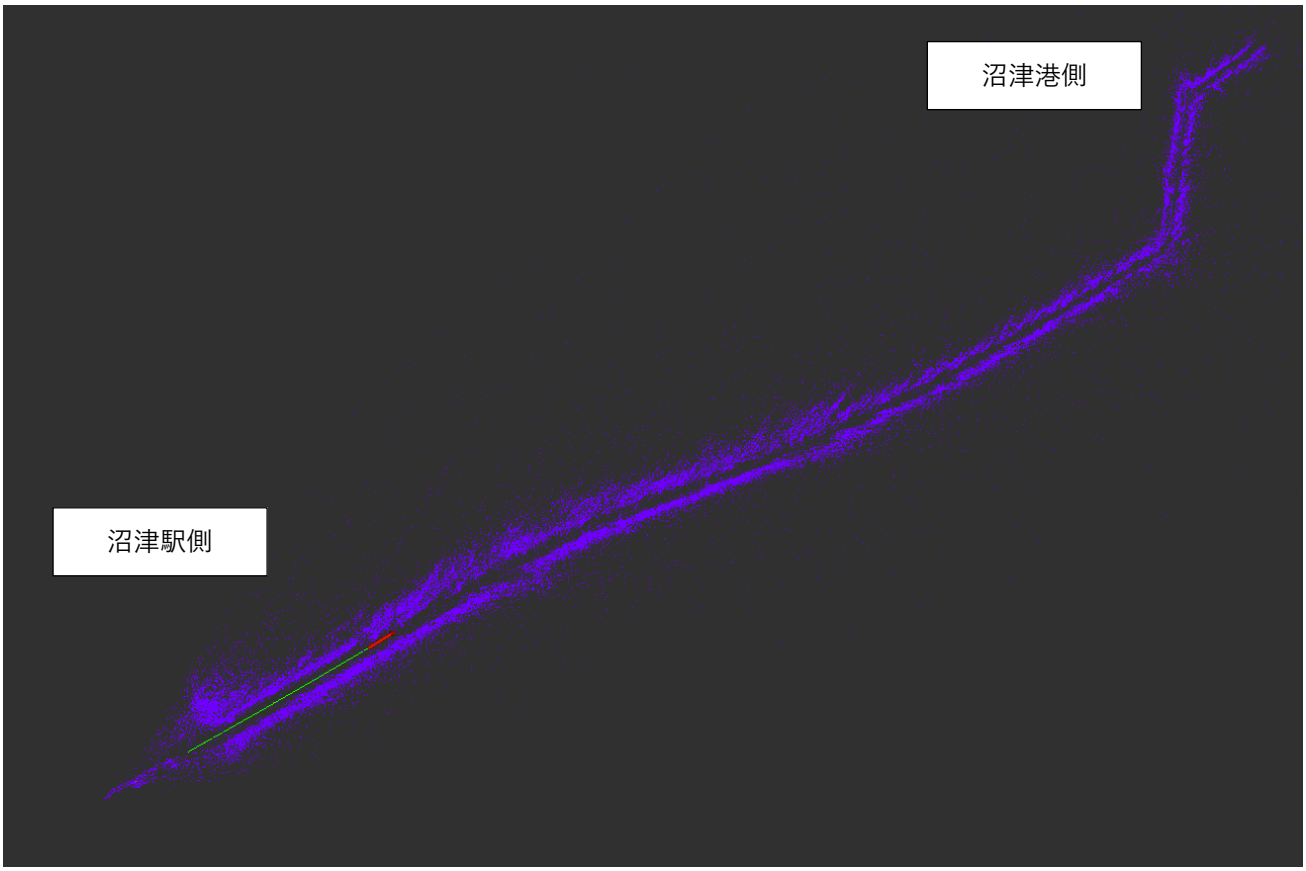


図 4-3 検証シナリオ 1、unity20231122-001.kdvm の VPS マップ点群表示

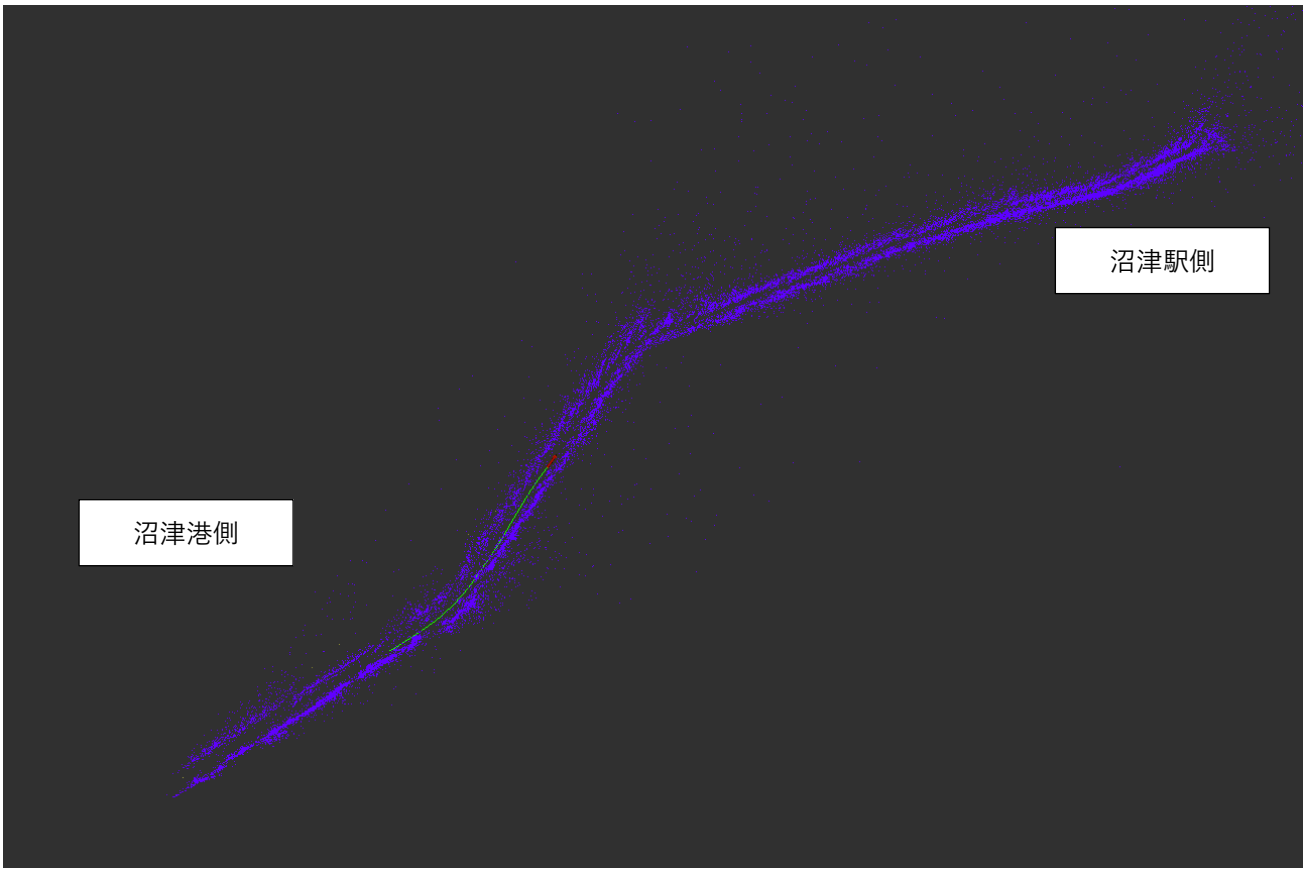


図 4-4 検証シナリオ 2、unity20231122-002.kdvm の VPS マップ点群表示

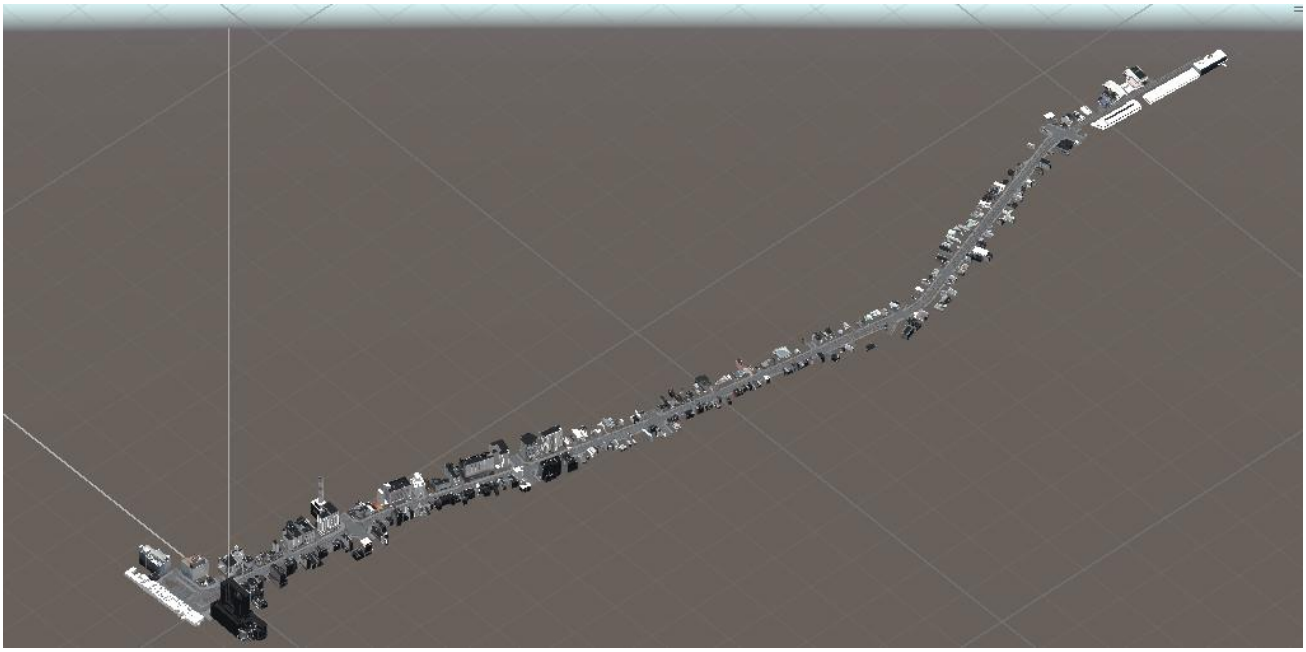


図 4-5 3D 都市モデルの対象エリア全域の様子

図 4-3、図 4-4 の青点は SLAM モジュールが認識した特徴点を表す。これは図 4-5 に示す対象ルートの概形をほぼ正確に反映している。本結果から、Unity による 3D 都市モデルをレンダリングした画像から KdVisual の VPS マップ作成は正常に行われたと考えられる。
また、これらの VPS マップをロードした状態の KdVisual に対して、Unity からのレンダリングした画像を入力したところ、正しく自己位置推定を行うことができたことから、VPS マップとしての最低限の要件は満たしていると言える。

4-2. 自己位置推定機能の検証

4-2-1. 検証目的

- 昨年度の C*のみの検証では、自己位置推定が長時間継続できない点が課題であった。今年度の検証では、新たに KdVisual を導入し 2 つの VPS システムを相補的に利用することで、3D 都市モデルを使用した VPS 用のマップを利用した C*と KdVisual による自己位置推定の動作確認と、両システムの座標の融合の動作確認、自己位置推定の精度、継続性、処理速度の向上を検証する。

4-2-2. KPI

表 4-4 KPI 一覧

No.	評価指標・KPI	目標値	目標値の設定理由	検証方法
1	自己位置推定の位置精度	最小値 0.5m 以下 全試行の 80%以上の試行で自己位置推定が 1m 以下でできていること	● スタンドアロンの GNSS の精度よりも一桁高い精度かつ動運転で要求される精度に近い数値として設定	RTK-GNSS 座標との差分
2	位置精度の改善動作確認	定量目標なし （段階的に真値の座標に近づく挙動となること）	● システムのフローとして精度の低い座標を各アルゴリズムにより段階的に高い精度にしていく形になっている。これが想定どおりに動作しているかを検証	RTK-GNSS 座標との差分
3	自己位置推定の継続性 （走行全体での評価）	全サンプル値の 75%以上で車両の走行に応じた閾値以内の差分であること	● 直前の座標値との差分が大きくなった場合、トラッキングが失敗しているとみなされるため、継続的にトラッキングできている指標として設定	トラッキングの分析
4	トラッキングの追従性	定量目標値なし （初期位置設定時の座標から車両の移動に応じた方向の変位が発生していること）	● 初期位置設定から車両の移動に推定座標が追従できていることを検証する指標として設定	トラッキングの分析
5	トラッキングの継続距離・時間	自己位置推定が連続して 30 秒程度、距離にして 150m 程度継	● 上記 4 変位の検証とともに、時間的にも継続してトラッキングできていることを検証す	30 秒に相当する連続サンプルで 4 の時系列差分が閾値（車両

		続して実行できること	る指標として 30 秒の継続時間を設定	速度から計算）を超えないこと
6	初期位置推定処理時間	初期位置推定処理が 2 秒以内に完了すること	● 初期位置推定のためのデータ入力から初期位置の出力までの時間。他の VPS が同様の処理をする時間を参考に同等の時間を設定	処理時間を計測する
7	自己位置推定処理時間	自己位置推定処理が 100 ミリ秒以内に完了すること	● C*について、1 回の自己位置推定処理にかかる時間として設定 ● （秒間 10 サンプルは必要と見られるため）	処理時間を計測する

4-2-3. 検証方法と検証シナリオ

- 検証方法

1) 自己位置推定の位置精度

自己位置推定結果座標のタイムスタンプに最も近いタイムスタンプの RTK-GNSS 座標との距離を計算した上で頻度分布を作成し、その最頻値や傾向を評価する。

x, y, z ：自己位置推定結果座標

$x_{RTK-GNSS}, y_{RTK-GNSS}, z_{RTK-GNSS}$ ：RTK-GNSS 座標

$$d = \sqrt{(x - x_{RTK-GNSS})^2 + (y - y_{RTK-GNSS})^2 + (z - z_{RTK-GNSS})^2}$$

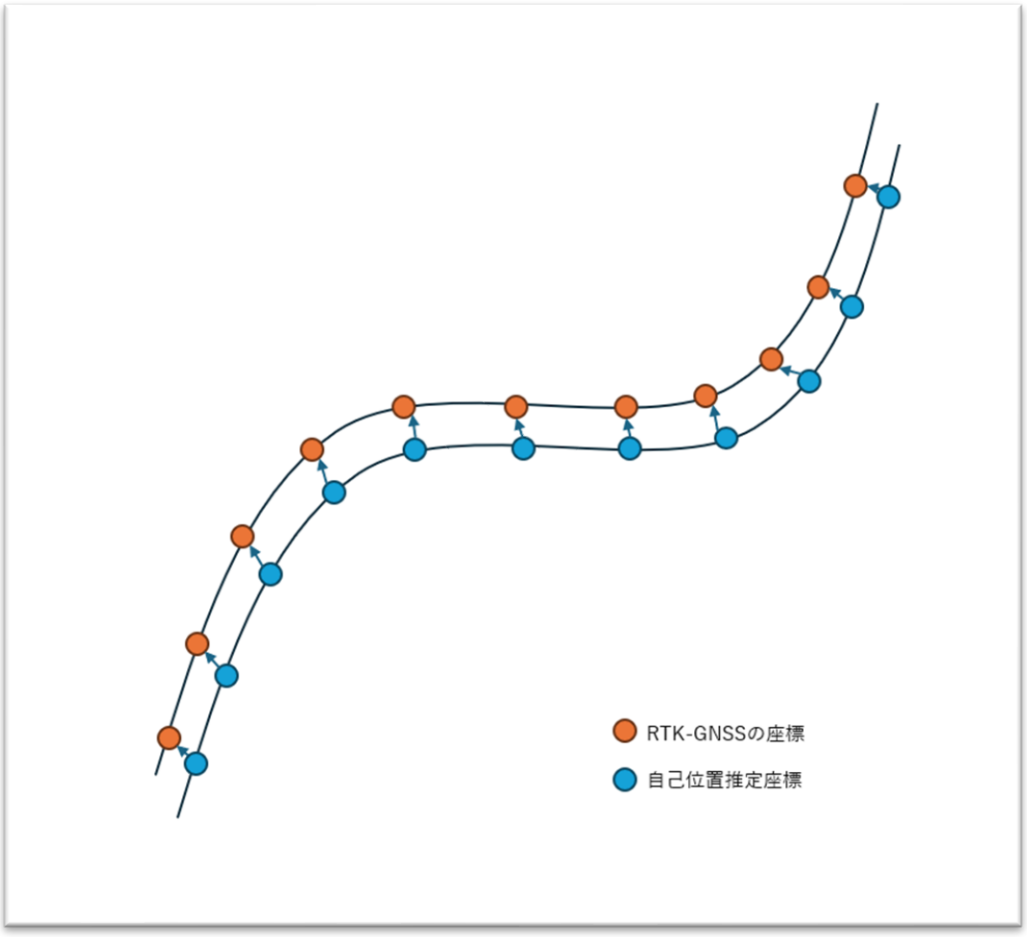


図 4-6 最近時刻の距離を計算

2) 位置精度の改善動作確認

前項の式を使い、スマートフォンで動作する初期位置入力プログラムで取得した GPS 座標、補正して C*に入力する座標、C*の自己位置推定結果座標、KdVisual の自己位置推定結果座標を、それぞれ RTK-GNSS 座標と比較し、順に精度が良くなっていくかの検証を行う。

3) 自己位置推定の継続性（走行全体での評価）

直前の自己位置推定座標との距離を計算する。図 4-7 左のように自己位置推定が定常的に動作し、継続的に自己位置を推定できていれば、この距離は移動体の速度を反映した、比較的滑らかな変化をすると考えられる。一方で、図 4-7 右のように移動体に追従できていない場合は、初期位置入力時のみ大きく位置が変わるため、断続的に大きな値が現れる挙動となる。そのため、本指標を計算することでトラッキングが正しく行われているかの検証を行う。

x_n, y_n, z_n ：n 番目の自己位置推定座標

$$d = \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2 + (z_n - z_{n-1})^2}$$

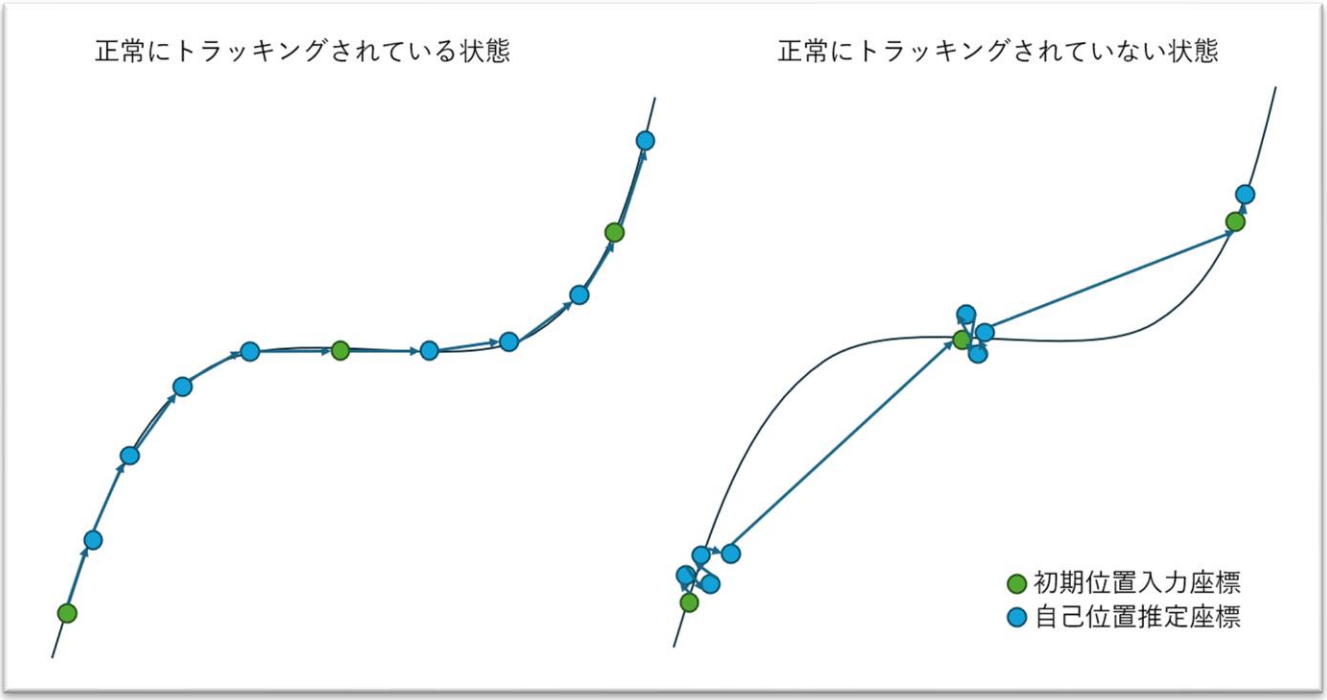


図 4-7 時系列差分解説図

4)　トラッキングの追従性（最終結果座標の各初期位置からの変位確認）

各初期位置入力を原点として、次の初期位置入力までの自己位置推定座標と初期位置入力時の自己位置推定座標との軸ごとの座標差分をとり、平面の座標値だけを散布図として表示する。これにより、図 4-7 右図の初期位置入力後の自己位置推定座標の挙動がどのような傾向を持っているかを分析する。移動体の移動に追従していれば順序に従って並ぶような分布をすると仮定できる。一方で追従できていなければランダムな分散に近い分布となると想定される。

x_n, y_n, z_n ：n 番目の自己位置推定座標
$x_{init}, y_{init}, z_{init}$ ：初期位置入力時の自己位置推定座標

$$dx = x_n - x_{init}$$
$$dy = y_n - y_{init}$$
$$dz = z_n - z_{init}$$

5)　トラッキングの継続距離・時間

3)の時系列差分が 30 秒間継続して閾値を超えないかを確認する。

6)　初期位置推定処理時間

初期位置推定処理にかかる時間をログで確認する。

7)　自己位置推定処理時間

自己位置推定処理にかかる時間をログで確認する。

- 検証ルート

自動運転車両の屋根に機材をセットアップした自動運転車で LOD3 の 3D 都市モデルが利用できる沼津市駅前から沼津港までの、約 2km の通称さんさん通りを往復し、システムの動作の確認と自己位置推定の検証を行う。
検証では、駅から港に向かう方向に合わせて作成した VPS マップと逆向きの方向に合わせて作成した VPS マップの 2 種類のマップを切り替えて使用する。
車両は東急株式会社所有の自動運転バスを使用する。本自動運転車両には名古屋大学開発の自動運転ソフトウェア「ADENU」が搭載されており、走行データをデータ分析に利用する。

表 4-5 検証ルート一覧

No.	使用した VPS マップ	ルート
1	unity20231122-001.kdvm	沼津駅前→沼津港
2	unity20231122-002.kdvm	沼津港→沼津駅前

- その他
  - 実施日
    - 2024 年 2 月 26 日

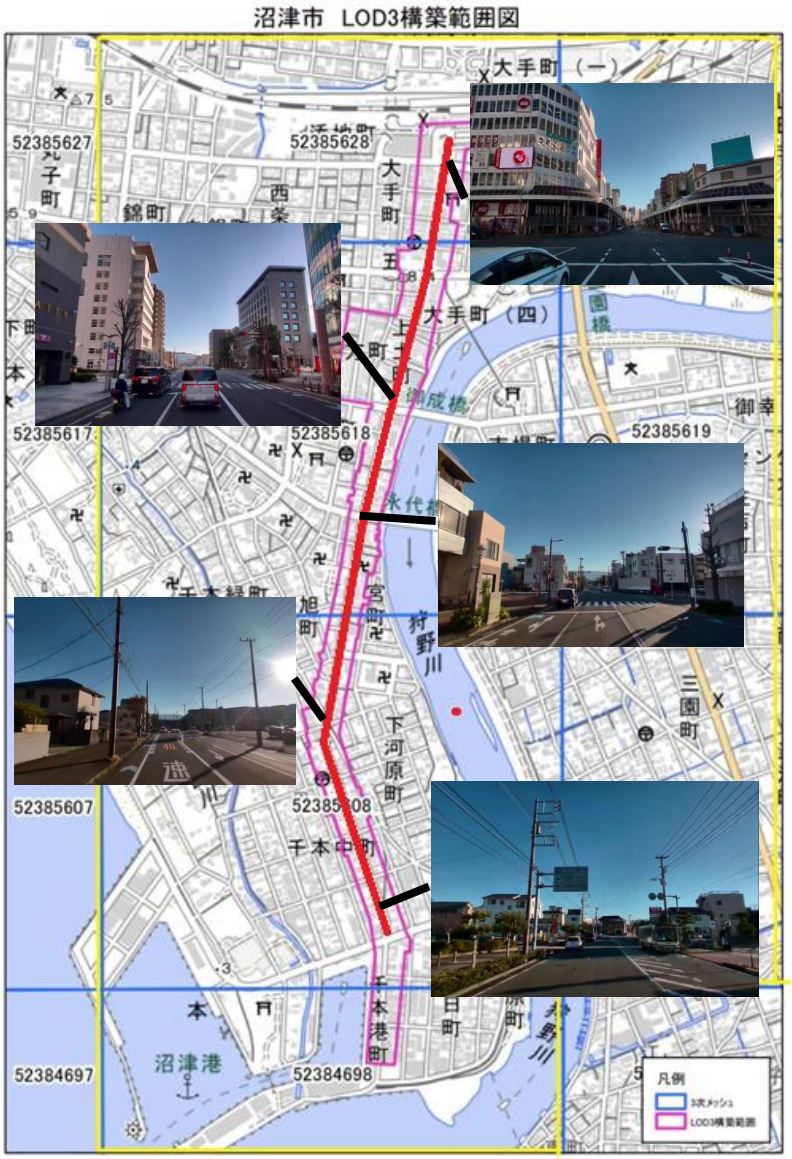


図 4-8 検証ルート（図内赤線）

4-2-4. 検証結果

- 検証結果まとめ
  - 本実証の結果、KdVisual は実走行で自己位置推定がほとんど成功しなかった。
    - 特徴点のマッチングが自己位置推定に十分な数成功していない
    - 誤った点をマッチングすることが多く、自己位置の誤判定が多い
  - C*に関しては昨年度同様の結果となったため、両手法を組み合わせて精度を向上させるという目標を達成できなかった。
  - C*は、今年度の改善として推測した座標をフィードバックして探索開始点の座標として使用しているが、自己位置推定に必要な範囲内にないため精度向上への寄与が認められなかった。

KdVisual の動作状況を分析すると、3D 都市モデルから作成した VPS マップと現実世界のカメラ画像との特徴点マッチングが出来ていないことが確認でき、またマッチングした場合でも誤った場所の特徴点同士を組み合わせている状況が見られた。この原因として、3D 都市モデルのレンダリング画像と現実世界のカメラ画像では、3D 都市モデルのテクスチャと現実世界における建物の側面外観の乖離や、3D 都市モデル上で再現されていない都市設備や植生などのディテールが異なることから特徴点ごとの特徴量の乖離が大きく、一致判定をすることが出来なかったと考えられる。「KdVisual」は画像内の特徴点に着目し周辺のピクセルの明度の差やパターンを分析し特徴量を算出している。そのため、レンダリング画像の特徴点としてメッシュのエッジやテクスチャのコントラストのはっきりした部分を抽出する傾向があるが、レンダリング画像と実写画像の比較では、明るさやコントラストの違いからマッチングに必要な類似性を見つけることが出来なかったと考えられる。一方で「C*」は局所的な類似性ではなく大域的な類似性をもとに判別することから、おおよそ似通った画像であればある程度の自己位置推定は可能となるため、この仕組みの違いが、今回の結果の差異につながったと言える。

表 4-6 検証結果サマリー

黄セル：KPI 達成	青セル：KPI 未達

検証内容	評価指標・KPI	目標値	結果		示唆
			項目	評価	
自己位置推定の座標誤差の評価	自己位置推定の位置精度	最小値 0.5m 以下 全試行の 80%以上の試行で自己位置推定が 1m 以下でできていること	検証ルート No.1 検証ルート No.2	未達	● C*は昨年度と同様の結果、KdVisual は自己位置推定をほとんどできなかった ● KdVisual は 3D 都市モデルの特徴点と実写の特徴点をマッチングできていない
各アルゴリズムの動作検証	位置精度の改善動作確認	段階的に真値の座標に近づく挙動となることこと	検証ルート No.1 検証ルート No.2	未達	● アルゴリズムによる座標の変位は確認できたが、精度の向上はなかった
自己位置推定の継続性の評価	自己位推定の継続性（走行全体での評価）	全サンプル値の 75%以上で車両の走行に応じた閾値以内の差分であること	検証ルート No.1 検証ルート No.2	未達	● 昨年同様、継続的な自己位置推定はできていない ● 推測した現在の座標値を入力しているが、推測に必要な範囲に入っていないと考えられる
	トラッキングの追従性	定量目標値なし（初期位置設定時の座標から車両の移動に応じた方向の変位が発生していること）	検証ルート No.1 検証ルート No.2	未達	● 昨年同様、継続的な自己位置推定はできていない ● 推測した現在の座標値を入力しているが、推測に必要な範囲に入っていないと考えられる
	トラッキングの継続距離・時間	自己位置推定が連続して 30 秒程度、距離にして 150m 程度継続して実行できること	検証ルート No.1 検証ルート No.2	未達	● 継続的な自己位置推定を連続して実行できていない ● 初期位置において自己位置推定が可能な範囲に入っていないと考えられる

自己位置推定の処理速度の評価	初期位置推定処理時間	初期位置推定処理が 2 秒以内に完了すること	検証ルート No.1 検証ルート No.2	達成	● 初期位置の取得自体は十分に速やかに完了した
	自己位置推定処理時間	自己位置推定処理が 100 ミリ秒以内に完了すること	検証ルート No.1 検証ルート No.2	未達	● 関わるコンポーネントが増えたこともあり、自己位置推定に時間がかかるようになった

最初に図 4-9 から図 4-12 に、検証で得られた自己位置推定座標と RTK-GNSS 及び ADENU が認識している自己位置座標の比較グラフを提示する。それぞれグラフの上が駅側、下が港側の位置関係を表した平面プロットである。RTK-GNSS と ADENU では大きな座標値の違いはなかった。また、KdVisual による自己位置推定ができなかったため、C*の結果のみ提示する。

おおむねルートに沿った座標はとれているが、一部で大きな誤差が生じている。初期位置入力の元になっているのがスマートフォン端末の GPS なので、GPS 衛星の配置によっては位置精度が大きく悪化することも考えられ、特に駅前のビルが多くあるエリアでの位置精度が悪くなっているのは、ビルによる遮蔽が原因ではないかと考えられる。

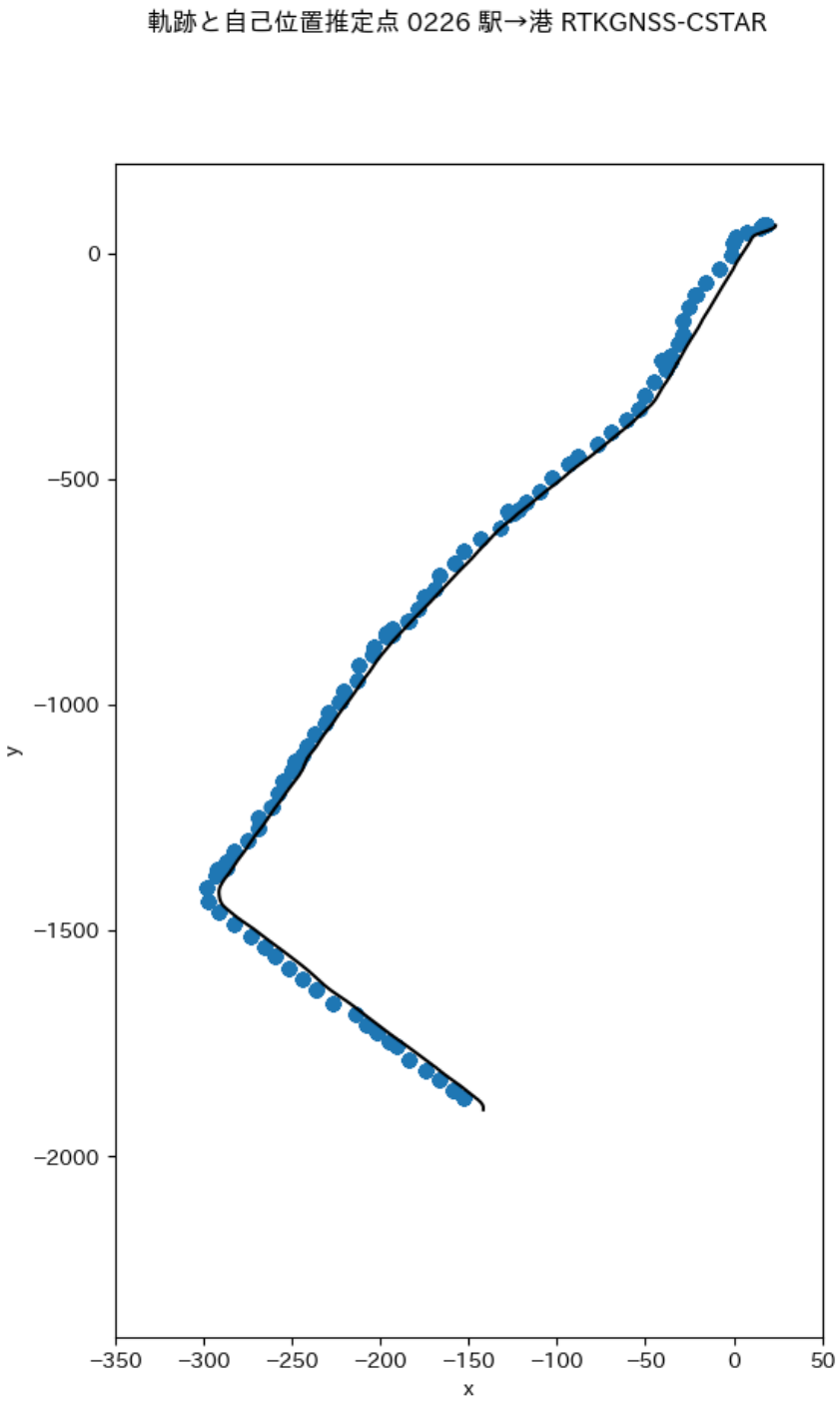


図 4-9 RTK-GNSS と C*の座標　検証シナリオ No.1

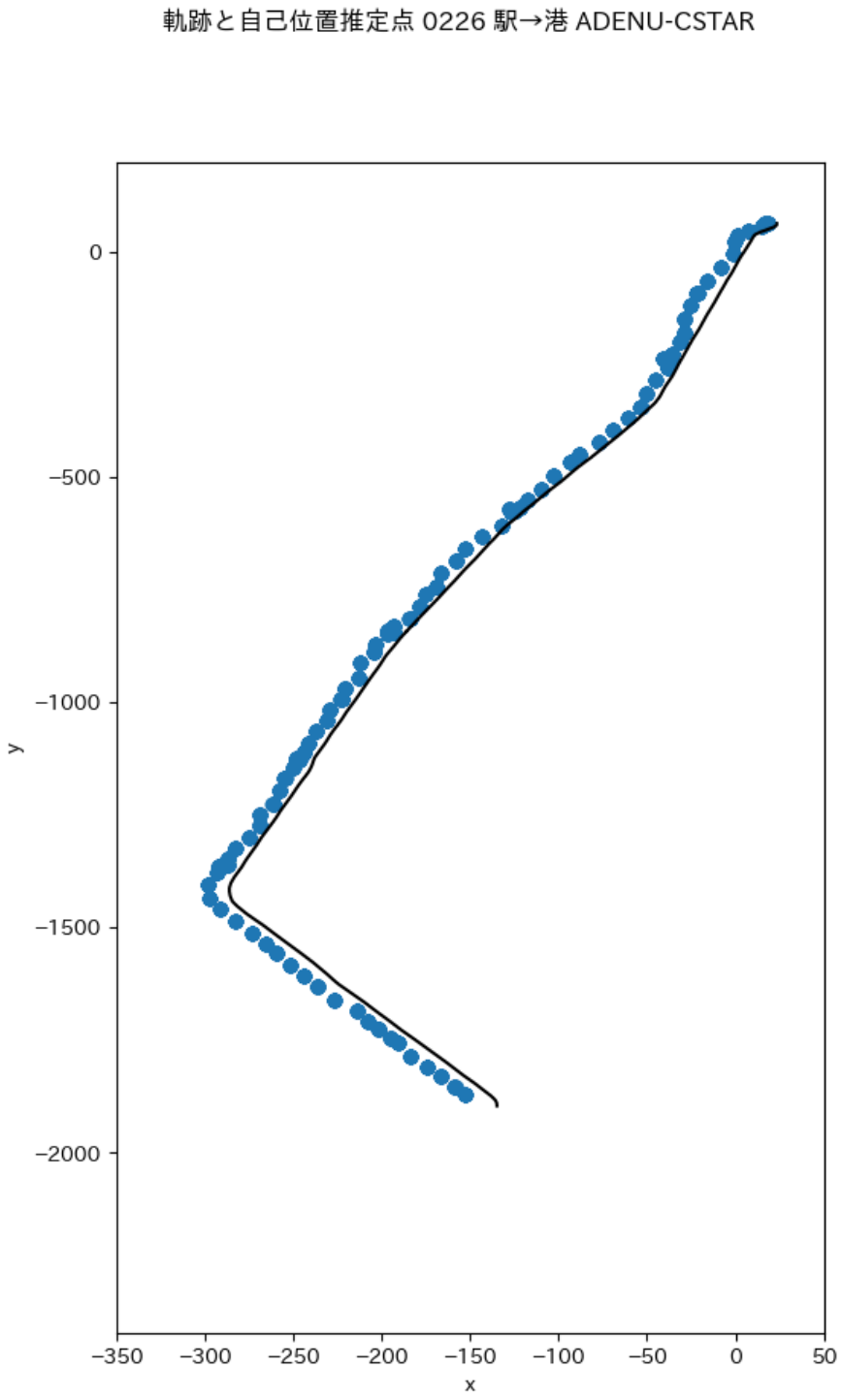


図 4-10 ADENU と C*の座標　検証シナリオ No.1

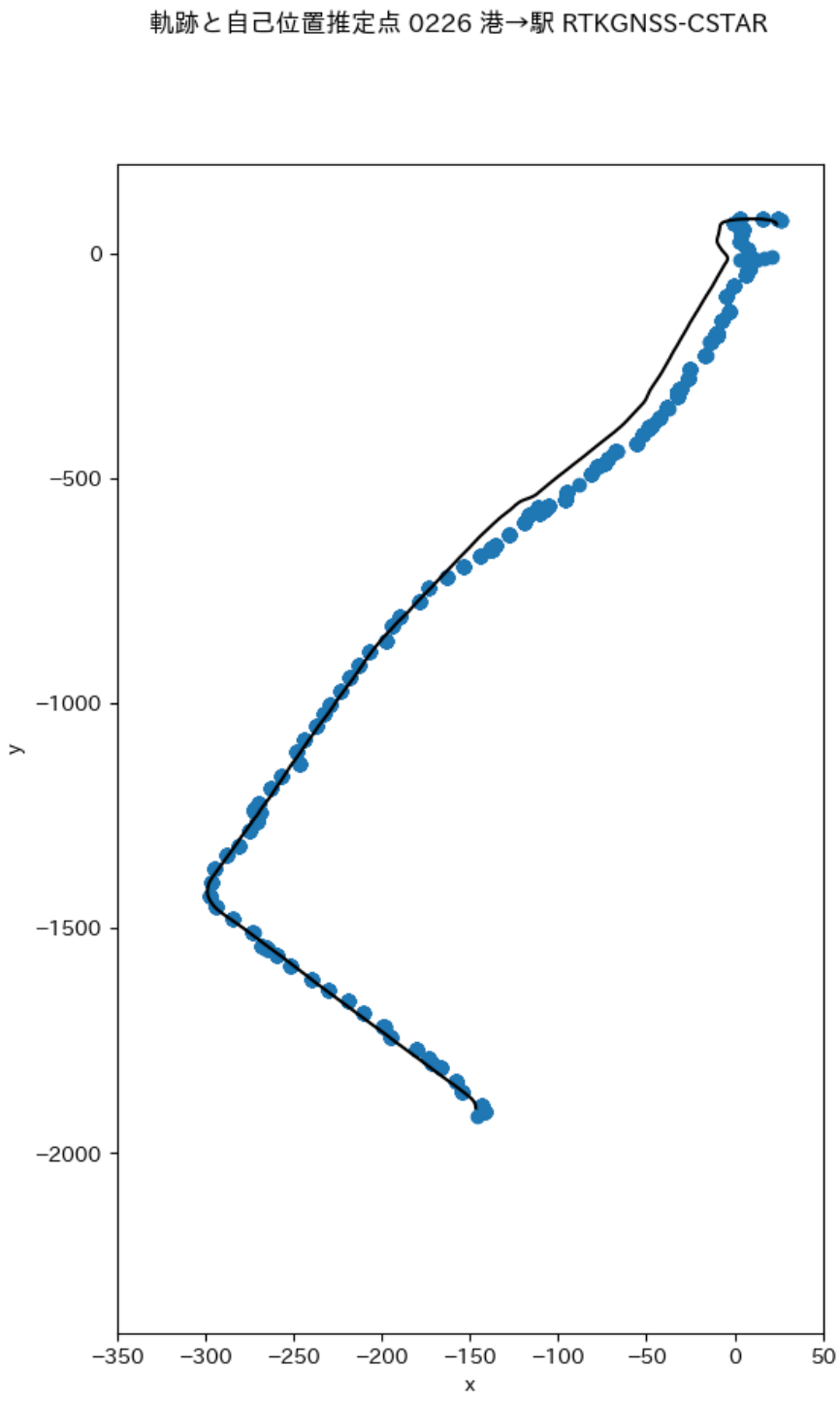


図 4-11 RTK-GNSS と C*の座標　検証シナリオ No.2

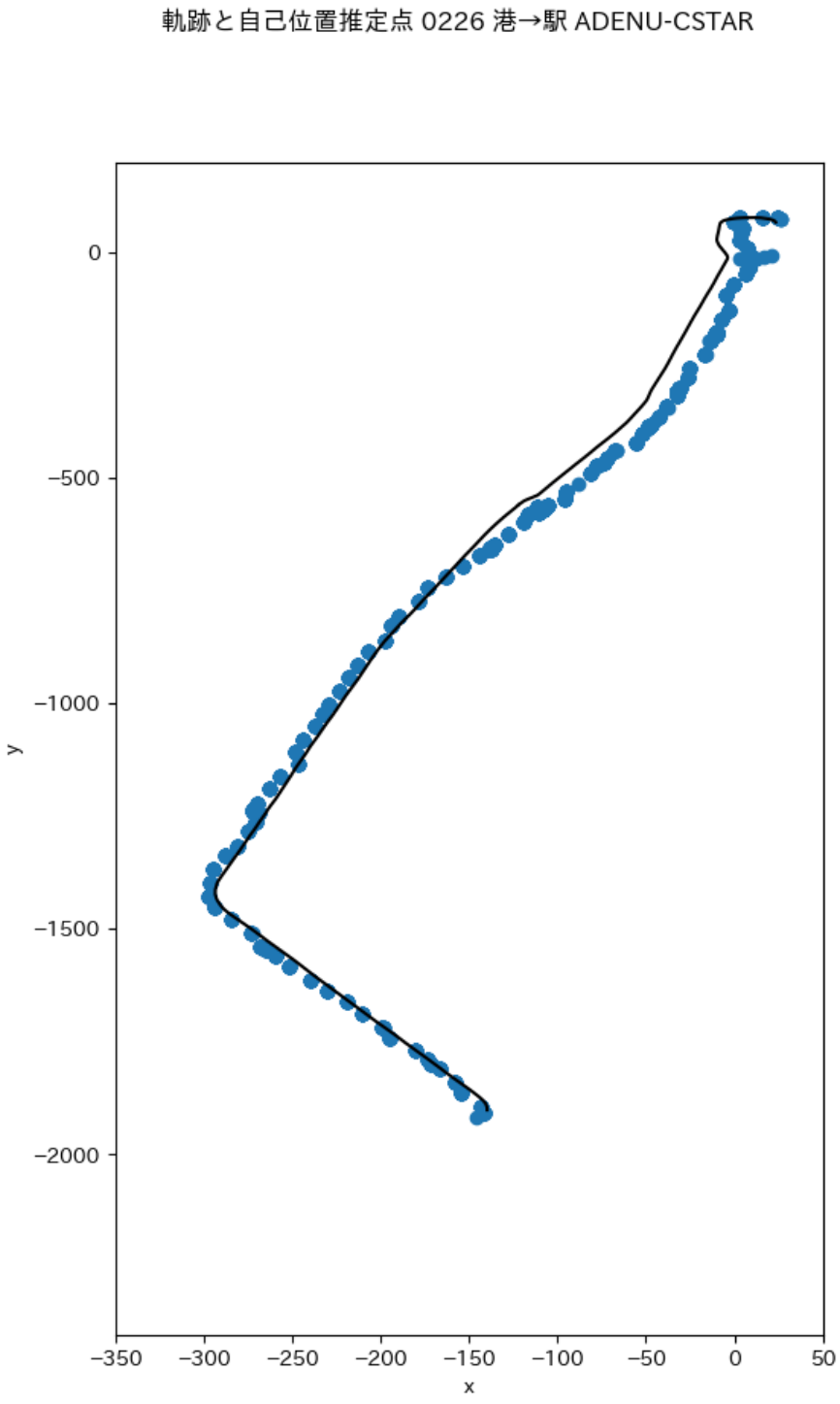


図 4-12 ADENU と C*の座標　検証シナリオ No.2

1) 自己位置推定の位置精度

- 検証結果まとめ
  - KdVisual は実写画像からの入力ではほとんどローカライズしなかった。また一部ローカライズした点でも、現在位置とは異なる誤った位置座標を出力した。このため、推定位置連携プログラムの動作を確認できなかった。
  - C*の自己位置推定は 5～15ｍの範囲にピークがあり、KPI 達成することはできなかった

- 検証結果詳細（C*の自己位置推定座標について）

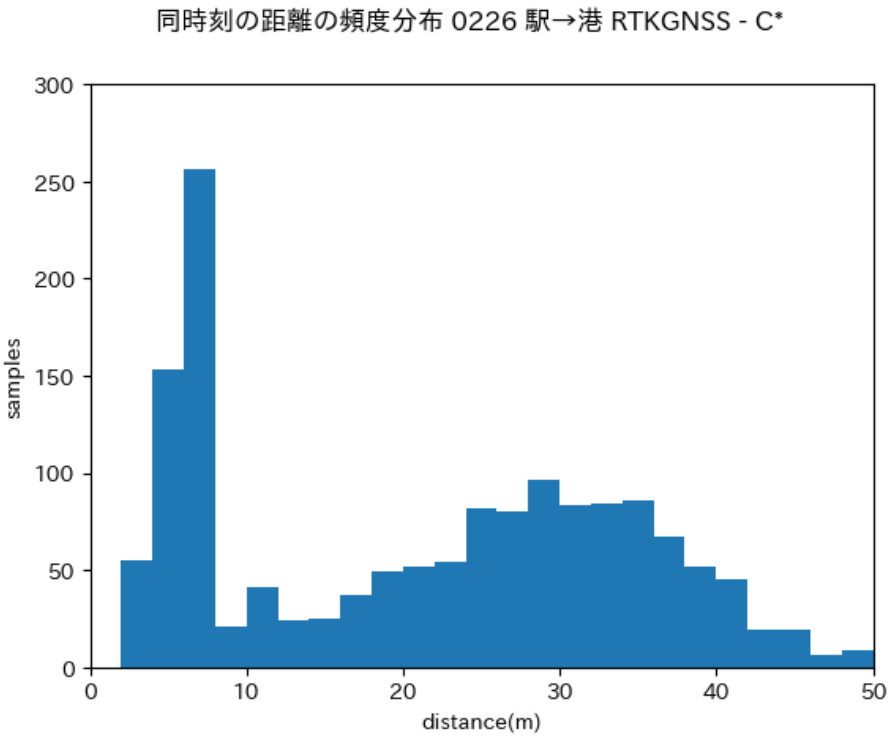


図 4-13 検証シナリオ 1 での C*と RTKGNSS 座標との距離の頻度分布

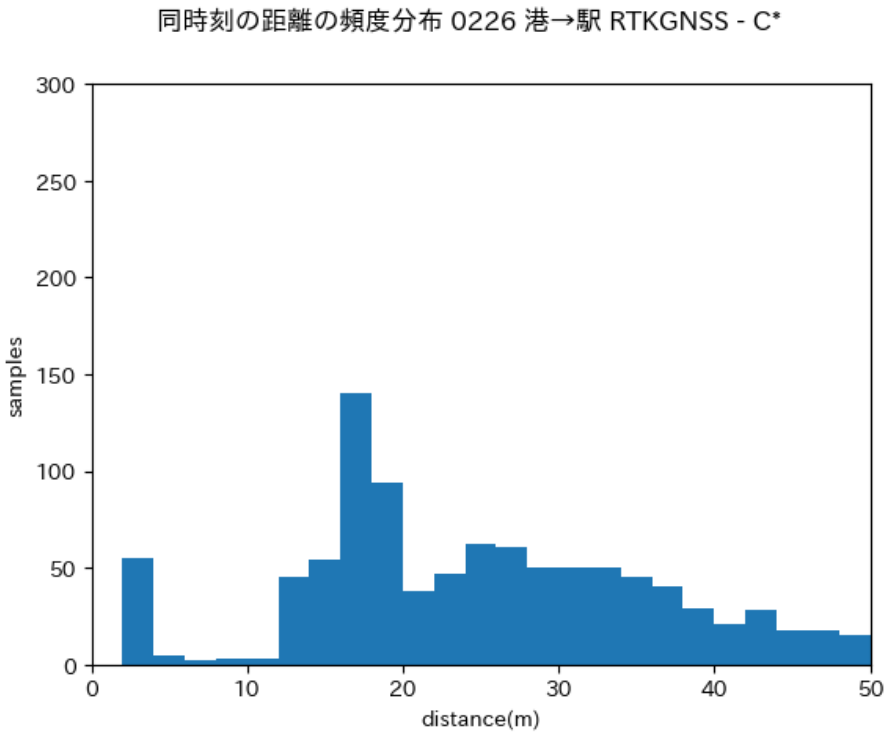


図 4-14 検証シナリオ 2 の C*と RTKGNSS 座標との距離の頻度分布

今回の結果は昨年度と同様の結果であり、大きな改善は見られなかった。

2)　位置精度の改善動作確認

- 検証結果まとめ
  - NID 探索による座標の変位は確認できたが、精度は変わらず KPI を達成できなかった。

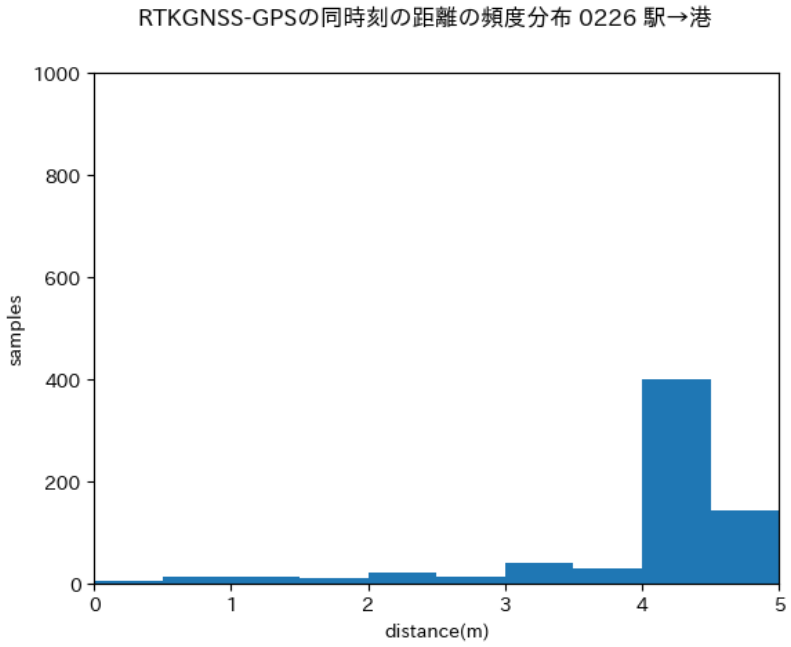


図 4-15 検証シナリオ 1 における RTK-GNSS とスマートフォン GPS の座標の距離の頻度分布

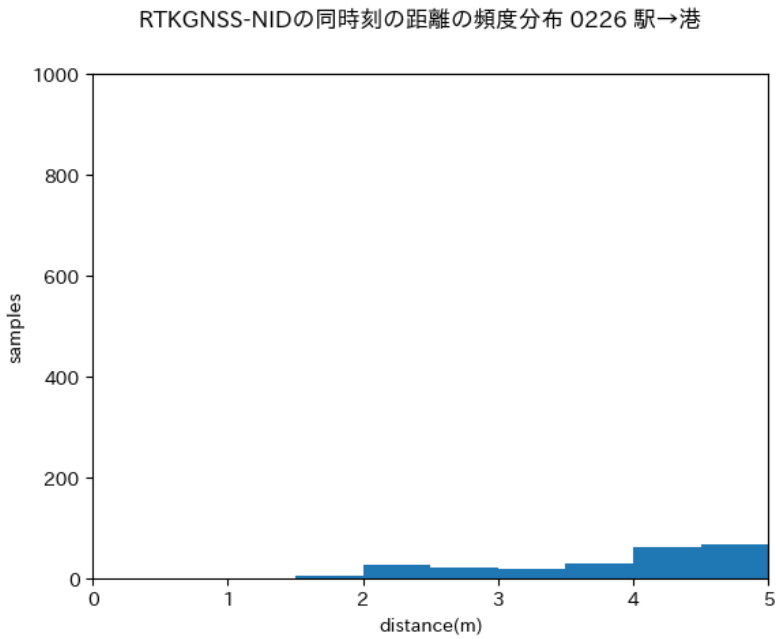


図 4-16 検証シナリオ 1 における RTK-GNSS と NID 探索結果座標の距離の頻度分布

図 4-15、図 4-16 の頻度分布の変化を見ると、スマートフォンの GPS で取得した座標から NID 探索結果座標が変位していることは確認できたが、昨年度同様大きな精度向上は認められなかった。

3) 自己位置推定の継続性（走行全体での評価）

- 検証結果まとめ
  - C*の挙動は昨年と同様に、初期位置入力時に断続的に差分が大きくなる挙動をしている。
  - トラッキング性能の改善は見られなかったため、KPI は達成できなかった。

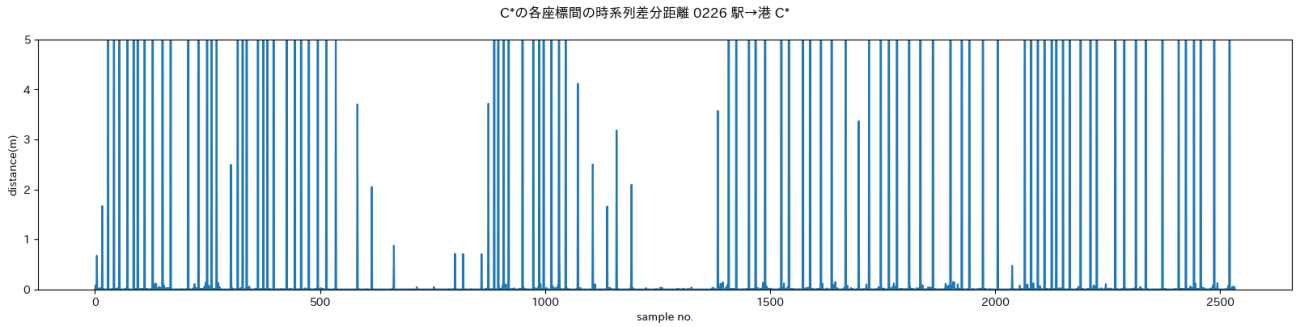


図 4-17 検証シナリオ 1 における C*座標の時系列差分

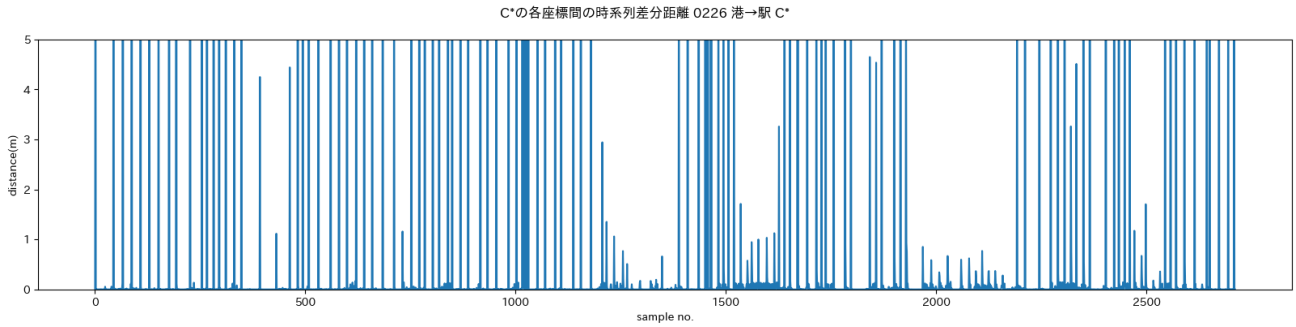


図 4-18 検証シナリオ 2 における C*座標の時系列差分

図 4-17、図 4-18 いずれでも 5 秒ごとに大きな差分が発生しており、初期位置入力時に大きく位置が動く挙動となっている。このため、昨年度同様に継続的な自己位置推定は成功しておらず、改善が見られなかった。

4) トラッキングの追従性（最終結果座標の各初期位置からの変位確認）

- 検証結果まとめ
  - C*は昨年同様に、初期位置周辺に分布する結果となり、トラッキングが成功していないことが確認できた。

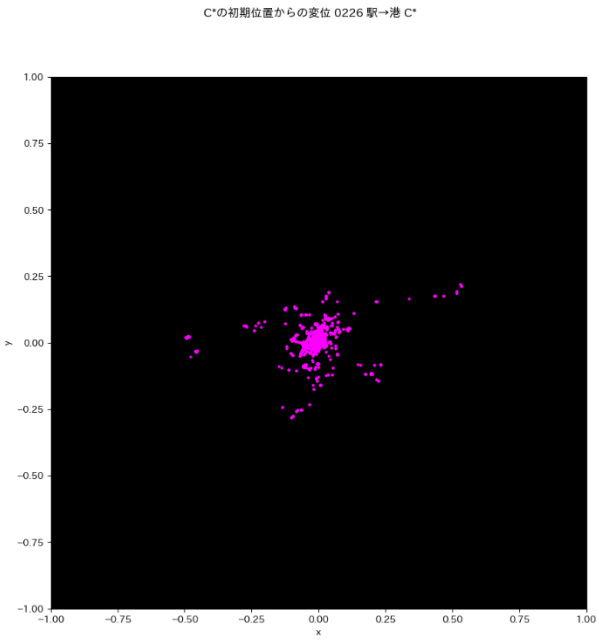


図 4-19 検証シナリオ 1 における C*座標の初期位置からの変位

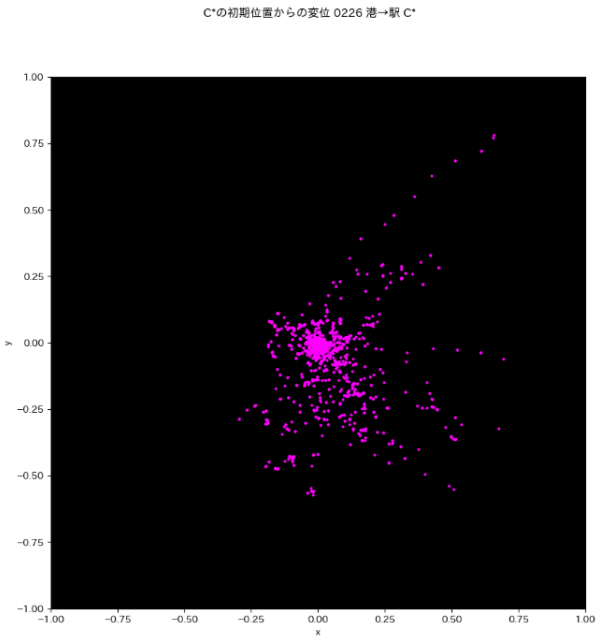


図 4-20 検証シナリオ 2 における C*座標の初期位置からの変位

5)　自己位置推定が連続して実行可能かどうか

- 検証結果まとめ
  - 3)のグラフから、トラッキングが成功しておらず、KPI は達成できなかった。

6)　初期位置推定処理にかかる時間

- 検証結果まとめ
  - ログから初期位置推定処理にかかる時間は平均 0.6 秒であり、KPI を達成することが出来た。

初期位置推定は C*においては探索範囲が増えるのみで通常の自己位置推定と処理自体は変わらないため、スマートフォンからの座標入力を加えても十分高速に動作した。

7)　自己位置推定処理にかかる時間

- 検証結果まとめ
  - ログから自己位置推定処理にかかる時間は平均 0.3 秒であり、KPI は達成できなかった。
  - 画像データの転送がボトルネックになっており、構成の見直しや最適化で高速化の余地はある。

前後の自己位置推定ログのタイムスタンプの差分をとり平均を計算することで、自己位置推定処理にかかる時間を計算した結果平均 0.3 秒であり、想定した時間よりも処理に時間がかかっていることが分かった。
原因としては、スマートフォン端末側の Ros-Sharp と VPS システムが動作している PC でスマートフォンからのデータ受信のために動作している rosbridge コンポーネントの内部での処理や、ネットワーク通信に時間がかかっており、そうした負荷処理時間を長くすると考えられる。
このため、今後は ROS に依存しない単体のプログラムのみで動作するような実装に変更することも考えられる。

5. 車両向け自律走行システム：BtoB ビジネスでの有用性検証

5-1. 検証目的

実証仮設に基づき、以下の検証目的を設定する。

【検証仮説】
3D 都市モデルを活用した VPS について、既存の LiDAR を利用した自己位置推定などと比較して簡便かつ安価な運用が可能となる。

主に以下の 2 点について、BtoB ビジネスに向けた有用性検証を行った。

- システムの運用における安定性・容易性
  - システムの動作の案転生、運用や設置の手軽さ容易性を評価する。
- システムのコスト比較
  - 既存のソリューションに対する優位性としてマップ作成や運用のコストを比較する。

5-2. 検証方法

- 検証方法

各指標については以下の検証を行う。

1)　システムの安定性
実証中の動作状況で所定時間の連続動作を確認する。

2)　運用の手軽さ、設置の容易性
実証実験の準備作業に必要な時間を計測する。

3)　実際に現地でマッピングするコストと 3D 都市モデルを使う場合のコストの比較
一般的な高精度マップ作成コストや、VPS マップ撮影コストと、本システムの 3D 都市モデル処理にかかる工数を比較する。

4)　使用する機材のコスト比較
一般的な自動運転機材のコストと本システムの現時点での構成のコストや、想定される実用化時のコストを比較する。

- 検証シナリオ

実際にシステムを動作させて確認する KPI については、4-13 の検証シナリオを実施する際に同時に計測する。コストの比較については、一般的な事例と本システムの現状の工数やコストを比較する。

5-3. 被験者

実際にシステムを稼働させた検証は、今回の実証実験の主担当であるホロラボ社員が実証を実施する過程で行った。

表 5-1 被験者リスト

分類	具体名称	部署	役職	担当業務	人数
事業者	ホロラボ	先進技術グループ	グループリード/ソフトウェアエンジニア	検証システムの開発・運用	1 名
			ソフトウェアエンジニア	検証システムの開発・運用	1 名

5-4. 検証の詳細

5-4-1. 検証項目と評価方法

表 5-2 検証項目と評価方法

検証観点	No.	検証項目	定量評価と目標値	目標値の設定理由	評価方法
システムの運用における安定性・容易性	1	システムの安定性	30 分間の連続動作が可能であること	実証実験を遂行するために必要な安定動作時間として設定	実証中の動作状況を記録する
	2	運用の手軽さ、設置の容易性	30 分ほどの作業で準備ができること	実証実験を遂行するために必要な作業時間として設定	作業に必要な時間を計測する
システムのコスト比較	3	実際に現地でマッピングするコストと 3D 都市モデルを使う場合のコストの比較	現地での作業によるマッピングと比べてコストを 1/10 以下に抑えること	MMS を使った高精度マップ作成コストや一般的な VPS マップ撮影作業の作業コストと比べて十分に低いコストとして設定	一般的な高精度マップ作成コストや、VPS マップ撮影コストと、本システムの 3D 都市モデル処理にかかる工数を比較する
	4	使用する機材のコスト比較	高度な自動運転機材と比べて機材コストを 1/10 以下に抑えること	現在実用化されている自動運転機材と比較して十分に低いコストとして設定	一般的な自動運転機材のコストと本システムの現時点での構成のコストや、想定される実用化時のコストを比較する

5-4-2. 実証実験の様子

実証実験において検証システムの設置や検証の様子を以下に示す。

図 5-1 機材設置中の様子

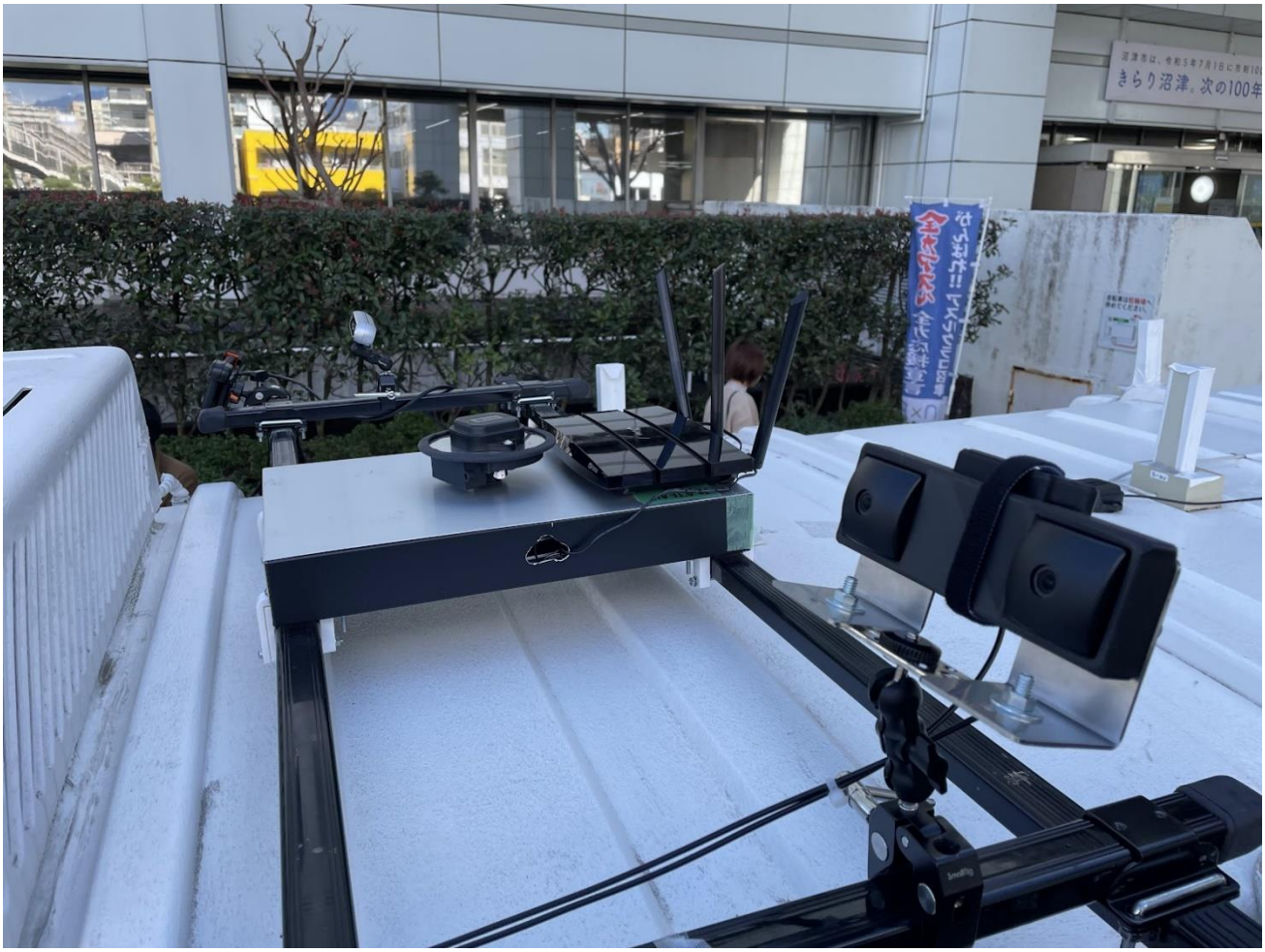


図 5-2 設置後の機材

図 5-3 実証走行の様子

図 5-4 実証走行の様子

図 5-5 システム稼働中の車内オペレーションの様子

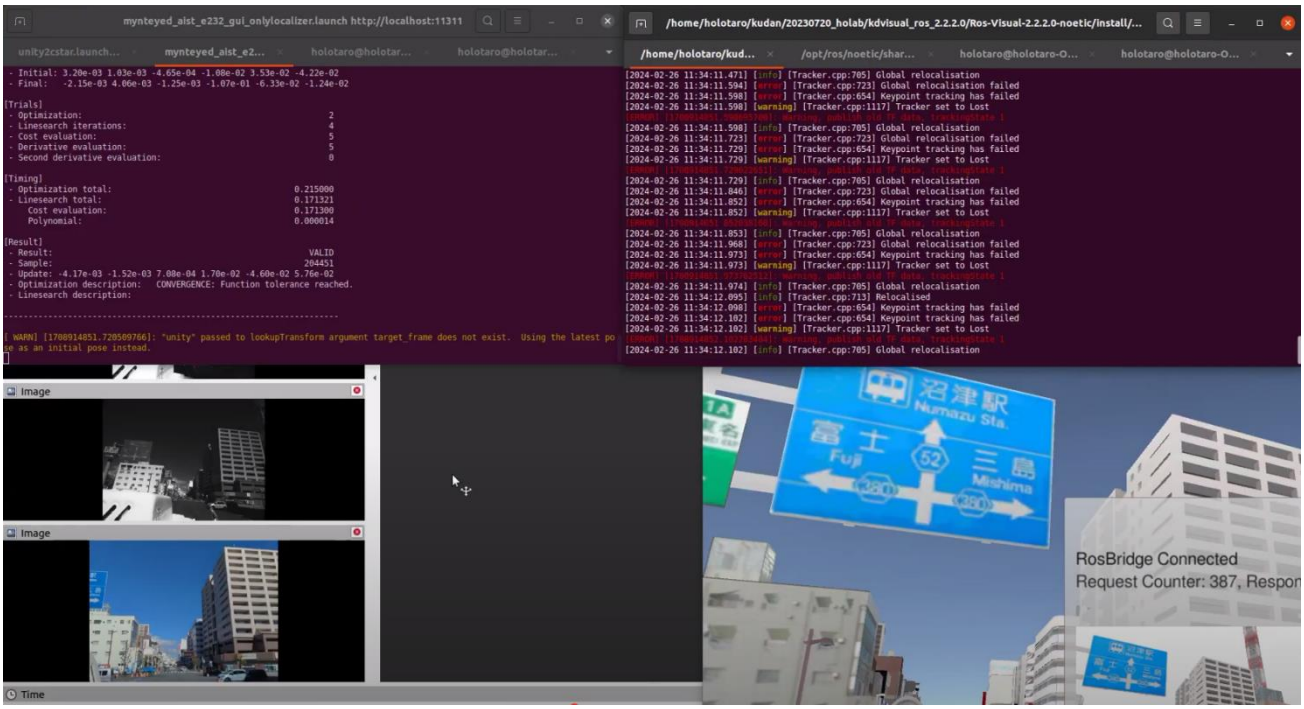


図 5-6 C*/KdVisual 動作画面

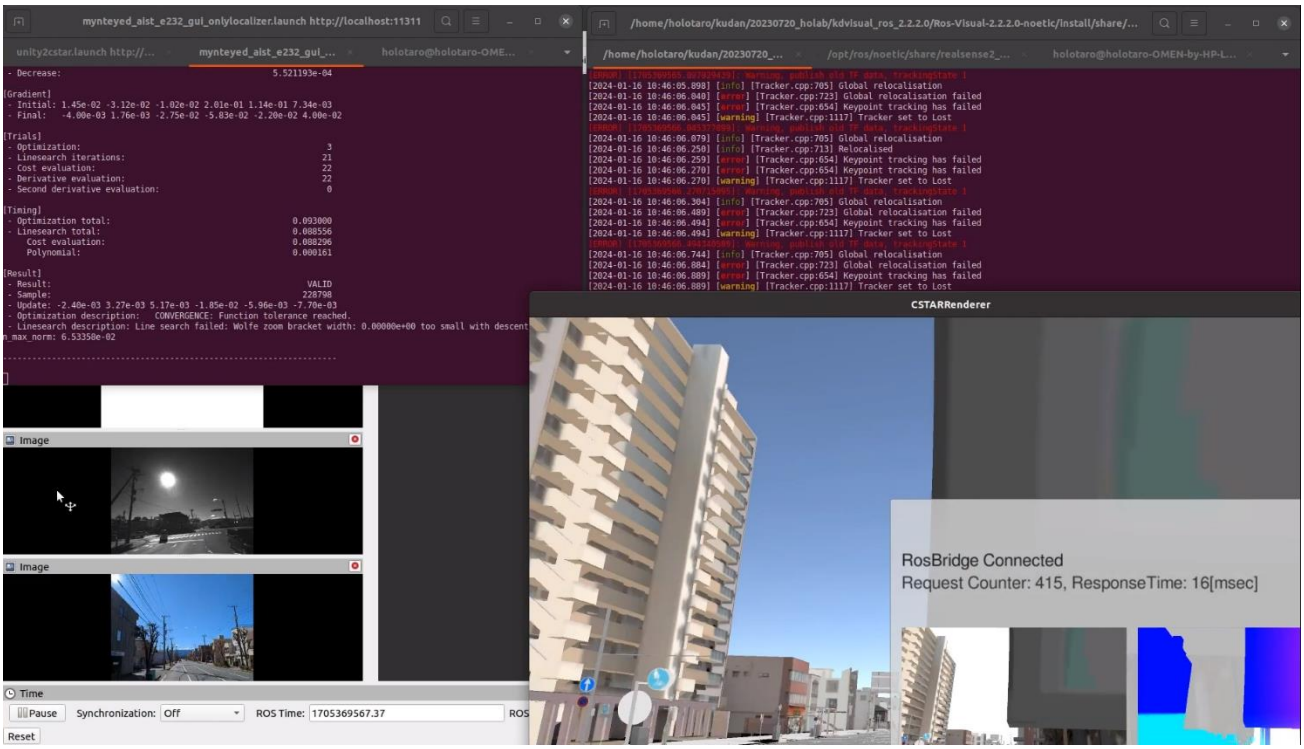


図 5-7 C*/Kdvisual 動作画面

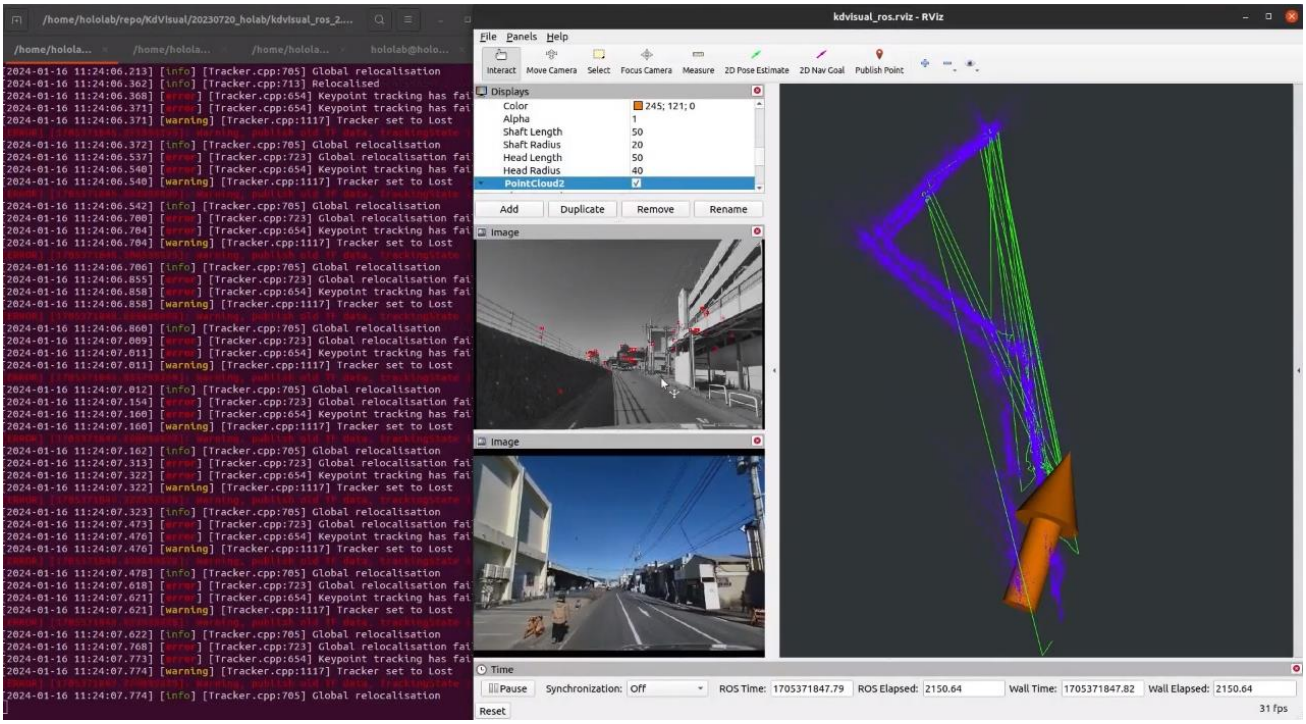


図 5-8 KdVisual 動作画面

図 5-9 初期位置入力機能動作画面

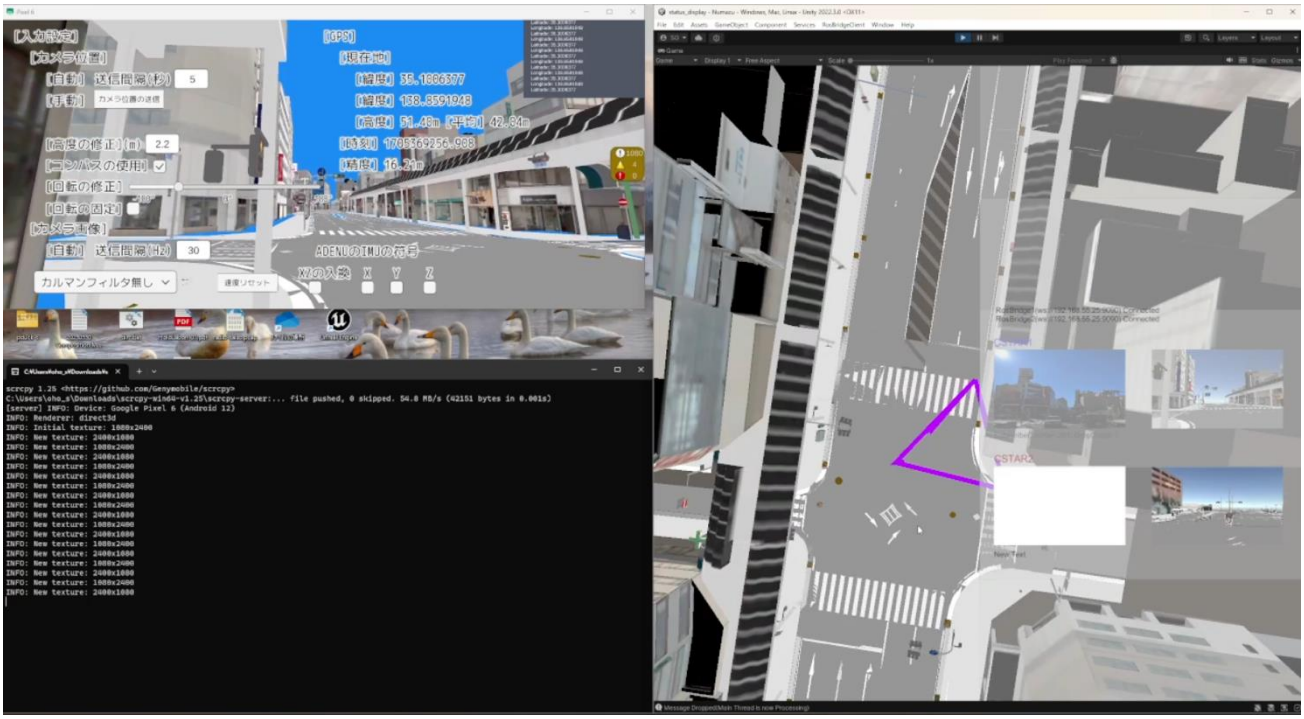


図 5-10 Status Display 動作画面

5-5. 検証結果

走行実証時の結果から実際にシステムを運用していく際の有用性を評価した。実運用に向けて現在のシステム構成と異なる可能性はあるが、今回の実証実験で活用したシステムでの結果として記載する。
ソフトウェア・ハードウェアともに安定性の面では目標を達成することができた。また、運用の手軽さという点においても簡易な作業で設置できる状態になった。
具体的に、利用データに関するコスト面では、オープンデータで公開されている 3D 都市モデルの優位が際立っており、作業面の効率化という観点においても現地に行く必要がなく手元の PC 上ですべてのマップ作成作業が完結することは大きな優位性となった。また機材コストは、既存システムは LiDAR のコストが大きく、カメラとコモディティ機器のみで動作する本システムに優位ではあった。
しかし、実運用時にはモビリティのシステム内に組み込んで使う想定のため、実運用のシステムとして使用するにはより多くの実証及び実績の積み重ねは必要になると考えられる。

1)　システムの安定性

- 検証結果まとめ
  - 実証中のルートの片道でおよそ 20 分の動作は複数回確認できた。
  - 連続動作として 1 時間以上の動作の確認も行った。
  - 今回の実証実験における KPI は達成することが出来た。

メモリリークの解消と処理の整理で昨年度の不具合は解消し、安定的にソフトウェアが動作するようになった。

2)　運用の手軽さ、設置の容易性

- 検証結果まとめ
  - 主に検証で使用した自家用車に設置する際は 30 分以内の作業で設置できた。
  - 自動運転車では、ネットワーク環境の違いの調整などに時間がかかったが、およそ 1 時間の作業で設置できた。
  - 今後構成を再考することで設置工数の削減は可能である。

ROS メッセージの形式で各コンポーネントを結合することで柔軟な構成変更も可能となり、セットアップも容易であった。自動運転車への設置では、事前に想定していたネットワーク設定と違っていたため、多少作業に時間がかかった。現在の構成は実証用のものであり、使用している機材等は汎用的なもののため、実用時にはより簡易な運用が可能な構成にしていくことはできると考えられる。

3) 実際に現地でマッピングするコストと 3D 都市モデルを使う場合のコストの比較

- 検証結果まとめ
  - 高精度マップとの比較
    - ✧ MMS による道路沿線のスキャンには最低でも数万円/km からのコストがかかる。
    - ✧ さらに高精度マップデータとするために、データ処理・変換などのコストがかかる。
    - ✧ 3D 都市モデルが利用できれば、すでに整備されていれば無料で利用することができ、コスト面での大きな優位性がある。
  - 既存の VPS との比較
    - ✧ 一般的な VPS 撮影では、現地での写真撮影が必須で、検証地域全域のマップを作成するためには 1 日～数日の作業が必要である。
    - ✧ 場合によっては、何度も現地に足を運び、マップの検証・調整が必要である。
    - ✧ 3D 都市モデルが利用できれば、3D レンダリング画像をもとにマップ作成可能となり、現地に行かずに VPS マップの作成が可能である。
    - ✧ マップ作成の自動化や検証と調整の繰り返しが容易である。

元データを用意するコストと、マップを作成する作業コストの両面で検証した。
一般的に自動運転に使われている高精度マップと比較すると、オープンデータである 3D 都市モデルは大きな優位性があるが、一方で作成されていない場所では利用できないため「すでに整備されている地域」では優位という結論となる。
マップを作成するコストは既存 VPS と比べると PC 上の作業のみで完結するため、大きな優位性がある。特に、マップ作成の試行錯誤が PC 内で繰り返し容易に行えるのは大きな利点であり、検証のために現地の映像が必要だがそれも録画データの活用が可能且つ繰り返して自動的に検証することも可能であり、ワークフローの大きな改善となる。

4) 使用する機材のコスト比較

- 検証結果まとめ
  - 自動運転用 LiDAR の価格は現状で低価格化が進んでいるとはいえ数十万円かつ複数台必要となる。
  - 今回使用したスマートフォン（Google Pixel 6）は定価 74,800 円、最新の高性能なものでも十数万円で購入可能である。
  - 今回使用したカメラは、TIER IV C1 が$800、Intel RealSense Depth Camera D455 が$239。
  - 高性能 PC は現状必要である。将来的にはスマートフォン単体や、シングルボードコンピューターを活用できる可能性もある。
  - コモディティ化したカメラ・コンピュータを使うことで廉価にシステムを構成できる。

機材コストは比較が難しいが、自動運転で使うクラスの LiDAR は現状でも数十万円以上の価格であり、それを複数台使用することを想定すると、カメラとスマートフォンを PC に追加するだけで利用可能な本システムはコスト優位性が高いと言える。さらに、KdVisual は安価な Single Board Computer でも性能を限定すれば動作するため、システムを改善することでより安価でコンパクトなシステムにできる可能性も考えられる。

表 5-3 機材のコスト比較

コンポーネント	一般的自動運転	本システム
センサ	LiDAR　数十万円×3 台	TIER IV C1 $800 Intel RealSense Depth Camera D455 $239 Google Pixel 6 74,800 円
PC	高性能 PC	高性能 PC

6. スマートフォン向けアプリケーション：実証実験の概要

6-1. 実証仮説

現状、AR ナビゲーションやラストワンマイル配送などの VPS 適用に向けて、開発前の空間マップ作成やそのメンテナンスが大きな課題となり技術進歩の大きな障壁となっている。今回の実証実験では、3D 都市モデルから作成した VPS 用の空間マップと現実空間でカメラから出力した点群を対応付け処理を行うことで自己位置推定を実現するアルゴリズムを構築し、AR ナビゲーションやラストワンマイル配送などコンシューマ向けサービスの開発において開発前の空間マップ作成を不要とし、メンテナンス負荷も低減することを目指す。

具体的にはプレティア・テクノロジーズ社が持つ VPS 用点群出力アルゴリズム（PretiaVPS）を活用し、実証対象エリアの 3D 都市モデルを Unity 上に読み込んで仮想空間上で点群を出力する。また、現実空間の同じ場所でスマートフォンのカメラから取得した点群と対応関係を持たせることで自己位置を導出する。

従来の PretiaVPS は、スマートフォンの 3D スキャナアプリで空間をスキャンして作成した点群を空間マップとし、別途スマートフォンから取得した画像情報を点群化して空間マップとマッチングさせることで自己位置を推定する仕組みとなっている。
これに対し、今回の実証実験では、3D 都市モデルを空間マップとして使用することで事前の 3D スキャンを不要とすることを目指した研究開発を行なった。

一方、VPS による自己位置推定において、空間マップ（点群）と自己位置推定時のカメラから取得した点群に大きな差分がある場合、適切に自己位置推定を実施できないという技術課題があった。3D 都市モデル（LOD3）は建物の形状や縮尺などは正確にとらえているものの、外壁のテクスチャは現実と異なる部分も多いほか、現実世界の環境は日々変化するため、単純に 3D 都市モデルを空間マップとして使用することはできない。
この課題を解決するため、異なるデータをもとにした点群と点群のマッチングアルゴリズム（点群 to 点群対応化アルゴリズム）を開発することが本実証における最も重要なポイントである。

これを実現することによって、3D 都市モデルが準備されている地域であれば簡易的に自己位置推定が可能なシステムを実現し、コンシューマ向けサービスとして、スマートフォンに本搭載、AR 技術を用いたアプリケーションを開発することで、VPS 技術を用いた観光やまちづくりなどへの活用用途が見出せる。

6-2. 実証フロー

今回の実証実験は下記７つのステップに分けて推進する。

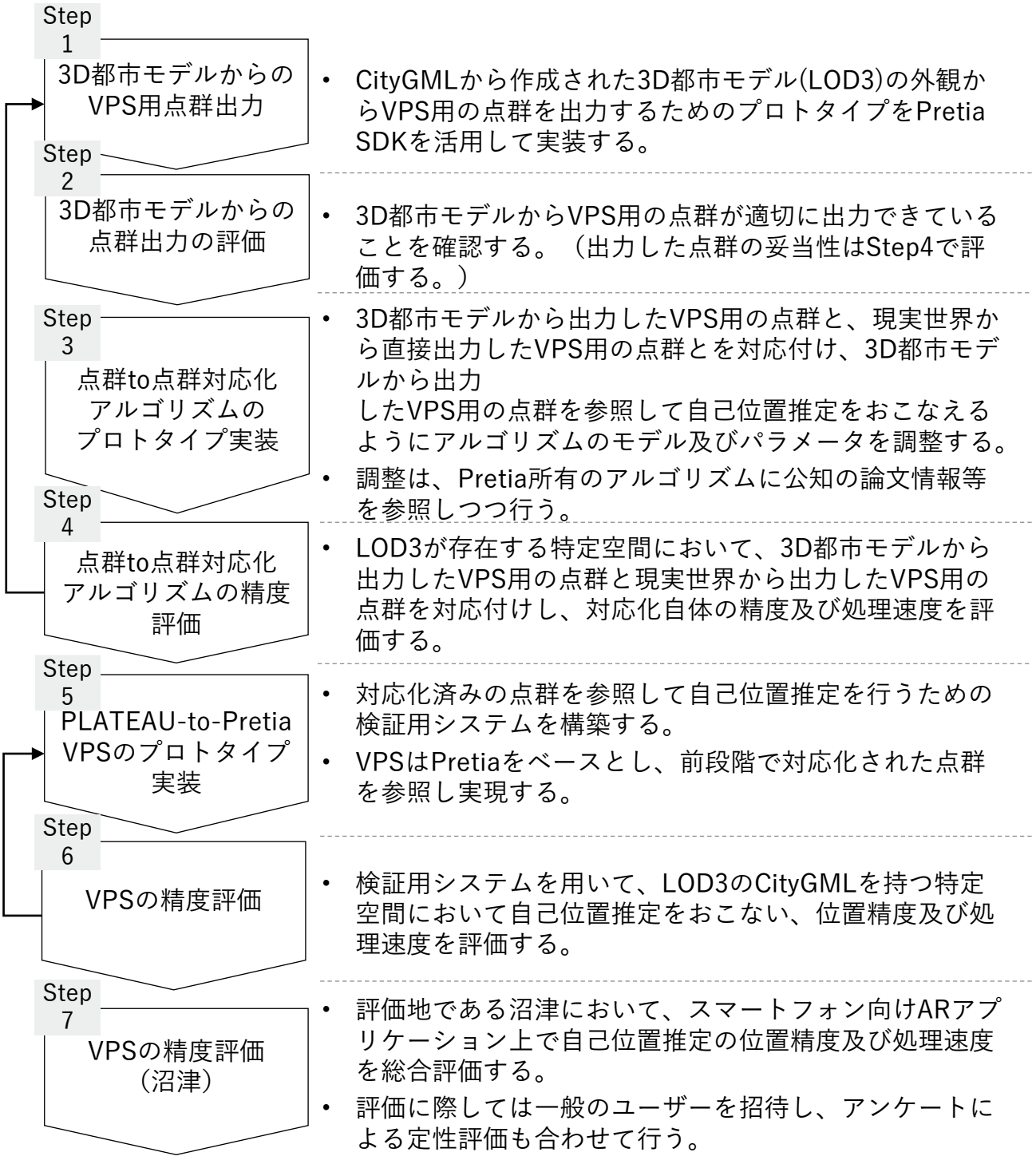


図 6-1 実証フロー

6-3. 検証ポイント

- 点群 to 点群対応化を活用した VPS アルゴリズムの確立
  - PretiaVPS を活用して 3 D 都市モデルから VPS マップ用の点群を出力し、現実世界のカメラ画像から取得した点群と比較して対応付けの処理を行うことで利用者の自己位置推定を可能とする VPS アルゴリズムの有用性を確認する。
- 上記で開発したアルゴリズムを活用した高精度なスマートフォン向け VPS システムの確立
  - コンシューマ向けアプリケーションへの実装を想定し、点群 to 点群対応化機能による高精度かつ高速処理を実現したスマートフォン向け VPS システムが確立できていることを確認する。

上記 2 点の検証ポイントについては、【8 章：実証技術の検証】にて検証結果を記載

- 今回の実証実験で開発した VPS システムによる自己位置推定機能を有したスマートフォン向け AR アプリケーションの有用性検証
  - 3 D 都市モデルを活用した VPS マップ及び今回開発した VPS による自己位置推定を搭載した AR アプリケーションが、GPS を活用した既存のナビゲーションアプリと比較して利用者の求める性能（位置精度及び処理速度）で利用できるのかを確認する。
  - また今度の社会実装を想定し、今回開発したアプリケーションをもとに利用者が求める VPS 技術へのニーズを抽出し、3D 都市モデルを活用した VPS 技術の機能向上に向けた改善項目を整理する。

上記 1 点の検証ポイントについては、【9 章：BtoB ビジネスでの有用性検証】にて検証結果を記載

6-4. 実施体制

表 6-1 実施体制

役割	主体	詳細
全体管理	国土交通省 都市局	プロジェクト全体ディレクション
	アクセンチュア	プロジェクト全体マネジメント
実施事業者	TOPPAN	ユースケース実証における企画・運営
	プレティア・テクノロジーズ	ユースケース実証における開発・検証

6-5. 実証エリア

表 6-2 実証エリア

項目	内容
実証地	静岡県沼津市
面積(ルート)	約 2km
マップ （対象エリアは赤枠内）	

6-6. スケジュール

表 6-3 スケジュール

実施事項	2023 年									2024 年		
	4 月	5 月	6 月	7 月	8 月	9 月	10 月	11 月	12 月	1 月	2 月	3 月
1. 実証計画の詳細化	⟷	⟷										
2. VPS 用アルゴリズムの研究開発		⟷	⟷	⟷	⟷	⟷	⟷	⟷				
2-1. 3D 都市モデルからの点群出力アルゴリズムの研究	⟷	⟷	⟷									
2-2. 点群 to 点群対応化アルゴリズムの研究		⟷	⟷	⟷	⟷	⟷	⟷	⟷				
2-3. モバイル-クラウドアーキテクチャのプロトタイプ開発					⟷	⟷	⟷	⟷				
3. 実証実験アプリケーション提供に向けたシステム統合							⟷	⟷				
4. 実証調査の実施								⟷	⟷	⟷		
5. 調査結果の分析										⟷	⟷	
6. 成果取りまとめ										⟷	⟷	⟷

7. スマートフォン向けアプリケーション：実証システム

7-1. アーキテクチャ

7-1-1. システムアーキテクチャ

本システムでは、PretiaVPS （プレティア・テクノロジーズ株式会社）をベースに、スマートフォン向けのアプリケーションとして利用可能な VPS 技術を研究開発することを目的とする。そのため、3D 都市モデルから生成した点群と、スマートフォンで撮影したカメラ画像からリアルタイムに生成した点群の双方を対応付けすることで自己位置推定を行うシステムを構築した。
具体的には、まず PLATEAU SDK for Unity を用いて CityGML 形式の 3D 都市モデルを読み込み、対象エリアの仮想空間を生成した。次に①Unity カメラで撮影して出力した PNG 形式の画像ファイルと、②スマートフォンで現実世界を撮影したカメラ画像をもとに、VPS 用の点群データに変換する技術である SfM（Structure from Motion）を用いて二種類の JSON ファイル形式の点群を出力した。次に、2 つ以上の点群を単一の座標系に統合する Point Cloud Registration 技術を応用し、二種類の点群をそれぞれ一定の領域で切り分け、切り分けられた点群の領域同士の位置合わせを行う ICP（iterative closest point）アルゴリズムを適用することで、双方の点群データに位置情報の対応関係を持たせた。ICP アルゴリズムは異なるデータセット間の三次元点群に対して位置合わせを行う際に用いるアルゴリズムであり、基準となるデータが持つ点群テータに対して入力された点群データを重ね合わせ、2 つの点群間の距離が最小となるためにはどの点同士を対応させるべきかということを判断し、マッチングすることが出来る。このように、カメラ画像をもとに生成した点群を再処理し、3D 都市モデルから生成した点群とマッチング可能とする機能を「点群 to 点群対応化」機能として開発した。

また、これらのシステムを活用するプラットフォームとして、一般ユーザーが利用するスマートフォン向けサービスを想定し、AR ナビゲーションアプリを開発した。具体的には、プレティア・テクノロジーズ株式会社が所有するクラウド上に「点群 to 点群対応化機能」を構築した上で、スマートフォンのカメラ画像から出力された点群の座標をリアルタイムに解析し、3D 都市モデル上の座標情報を取得することで利用者の自己位置推定が可能なシステムとなっている。この自己位置推定された利用者の位置情報をもとに、利用者がスマートフォンで撮影している画面に対して、店舗のポップアップや移動ルートを表示する AR ナビゲーションアプリとして構築することで VPS 技術の有用性を検証できる仕様を整えた。

本システムのシステム・アーキテクチャは下図の通りである。

本システムのシステム・アーキテクチャは下図の通りである。

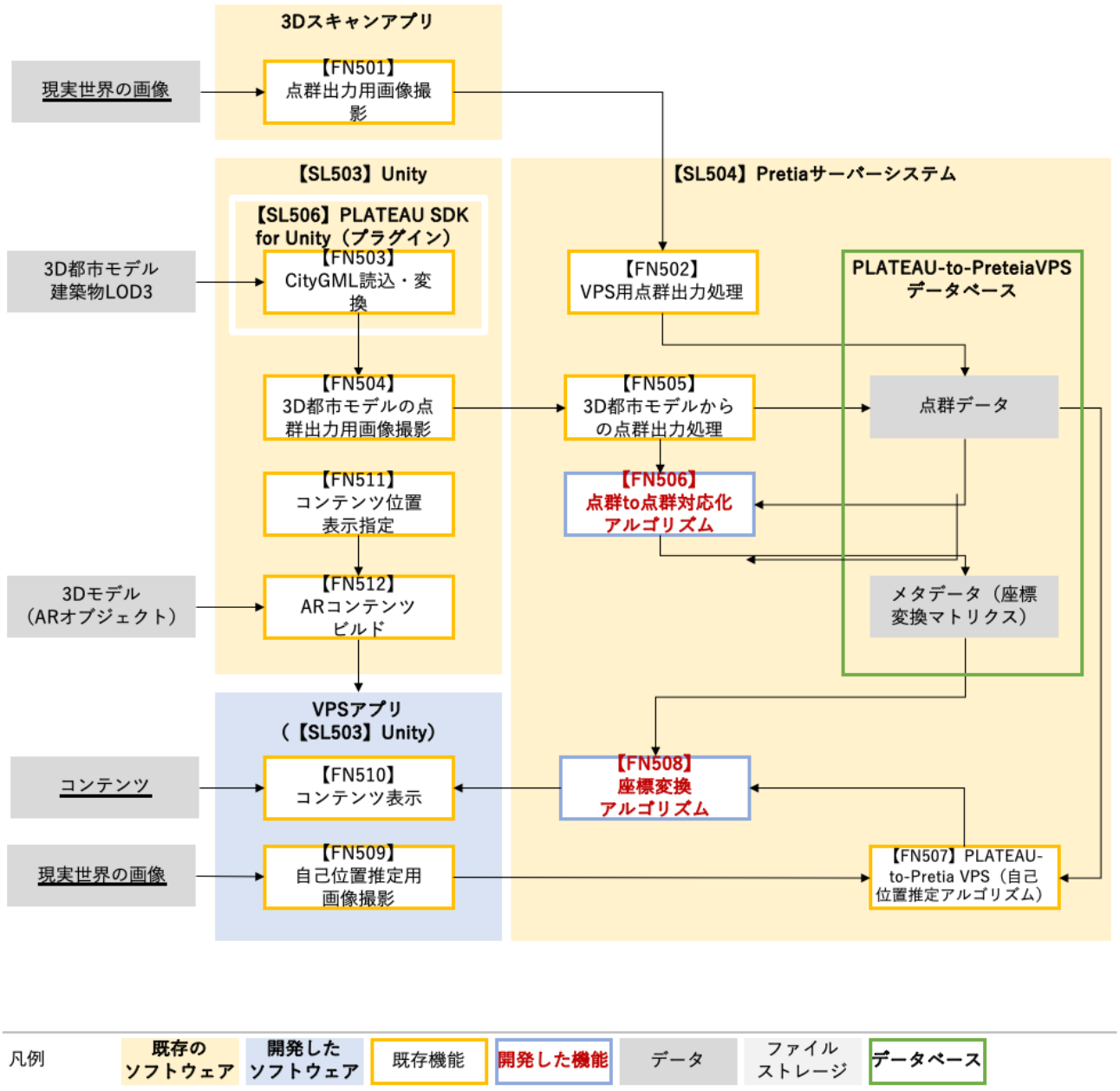


図 7-1 システムアーキテクチャ

7-1-2. データアーキテクチャ

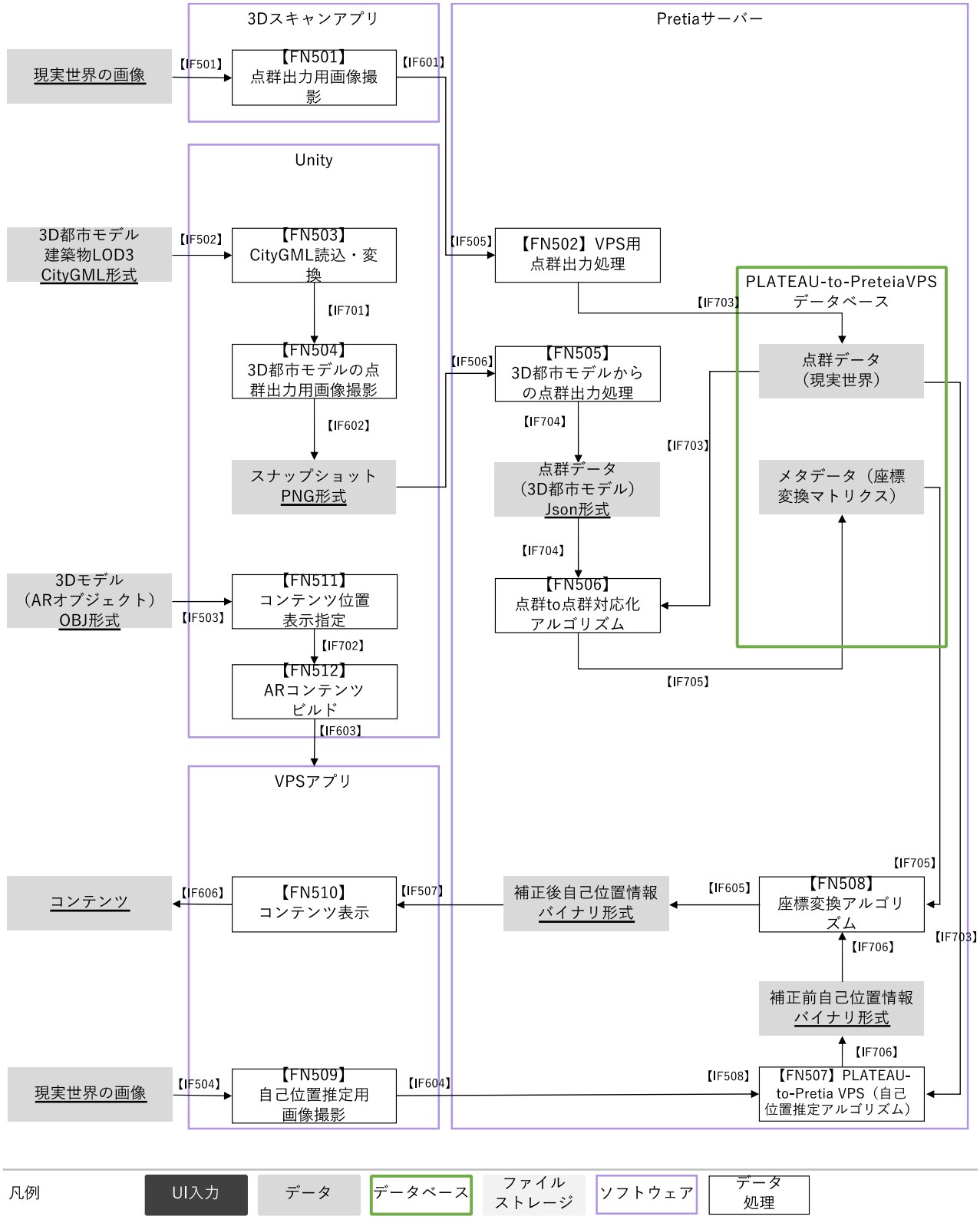


図 7-2 データアーキテクチャ

7-1-3. ハードウェアアーキテクチャ

7-1-3-a. 利用したハードウェア一覧

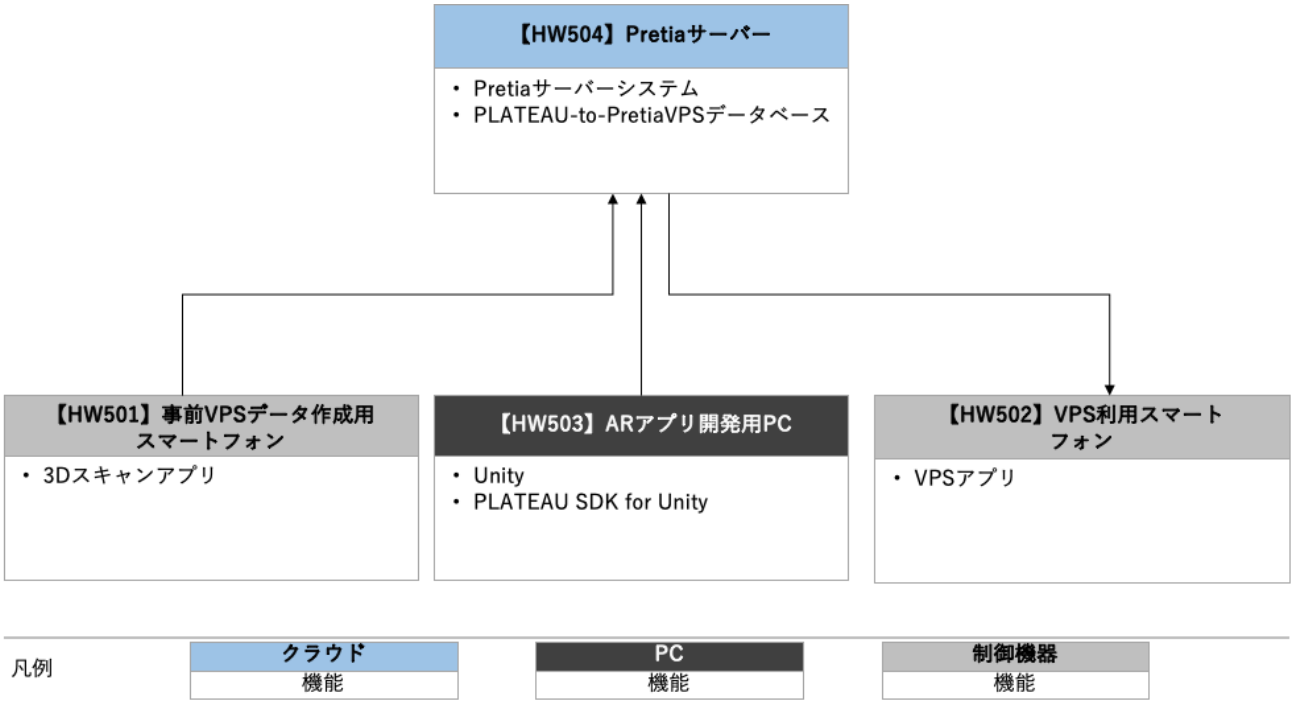


図 7-3 ハードウェアアーキテクチャ

表 7-1 利用したハードウェア一覧

ID	種別	品番	用途
HW501	事前 VPS データ作成用スマートフォン	iPhone12 pro	● テストフィールドの点群データ作成のためのスキャンデータ入力
HW502	VPS 利用スマートフォン	iPhone 12 mini	● GPS による位置情報の入力 ● カメラからの映像入力 ● AR コンテンツの表示
HW503	AR アプリ開発用 PC	Macbook pro	● Unity による AR アプリケーションの開発
HW504	Pretia サーバ	-	● プレティア・テクノロジーズ社（以降 Pretia）が提供する VPS を稼働させるサーバ

7-1-3-b. 利用したハードウェア詳細

1) 【HW501】事前 VPS データ作成用スマートフォン：iPhone12 Pro

- 選定理由
  - LiDAR を搭載している（本プロジェクトで直接利用するシステムとして LiDAR センサは不要だが、点群出力アルゴリズムにおける処理の妥当性を確認するため、LiDAR 搭載機種を選定）

- 仕様・スペック
  - サイズ:71.5×146.7×7.4mm
  - 重量: 187g
  - データ容量 256GB
  - センサ: LiDAR スキャナ、 3 軸ジャイロ、加速度センサなど
  - 位置情報: GPS, GLONASS, Galileo, QZSS, BeiDou

- イメージ

図 7-4 iPhone12 Pro[8]

[8] 公式 HP より抜粋：https://support.apple.com/kb/SP831?locale=ja_JP

2)　【HW502】VPS 利用スマートフォン：iPhone 12 mini

- 選定理由
  - 必要十分かつ市場で一般的に利用されている端末であるため
- 仕様・スペック
  - サイズ:64.2×131.5×7.4mm
  - 重量: 133g
  - データ容量 256GB
  - センサ: 3 軸ジャイロ、加速度センサなど
  - 位置情報: GPS, GLONASS, Galileo, QZSS, BeiDou
- イメージ

図 7-5 iPone12 mini[9]

[9] 公式 HP より抜粋：https://support.apple.com/kb/SP829?locale=ja_JP

3) 【HW503】AR アプリ開発用 PC：MacBookPro

- 選定理由
  - 必要十分な性能
  - 持ち運び、実証実験中の取り回しのしやすさ
- 仕様・スペック
  - CPU：Apple M2 チップ
  - メモリー: 16GB(8GB ユニファイドメモリ)
  - SSD: 512GB
- イメージ

図 7-6 MacBook Pro[10]

[10] 公式 HP より抜粋：https://support.apple.com/kb/SP870?locale=ja_JP

4) 【HW504】Pretia サーバ

- 選定理由
  - Pretia が持つ VPS 機能を利用するため
- 仕様・スペック
  - 非公開
- イメージ
  - クラウド上の専用サーバであり、イメージなし。

7-2. システム機能

7-2-1. システム機能一覧

1)　機能一覧

表 7-2 機能一覧

※赤文字：既存改修・新規開発

大分類	小分類	ID	機能名	機能説明
マップ作成機能	現実世界からの点群出力	FN501	点群出力用画像撮影	● VPS 用点群の作成に必要な画像を撮影し Pretia サーバへ送信する
		FN502	VPS 用点群出力処理	● カメラ画像として取得した現実世界の情報を VPS 用の点群データ①に変換する
	3D 都市モデルからの点群出力	FN503	CityGML の読込・変換	● PLATEAU SDK により CityGML を OBJ ファイルへ変換し Unity 上の開発環境へ取り込む
		FN504	3D 都市モデルの点群出力用画像撮影	● 点群出力の前処理として、3D 都市モデルを Unity 上のシミュレーションカメラで撮影し PNG ファイルへ変換する
		FN505	3D 都市モデルからの点群出力処理	● PLATEAU の 3D 都市モデルの情報を VPS 用の点群データ②に変換する
座標変換マトリクスの作成		FN506	点群 to 点群対応化アルゴリズム	● 現実世界から出力した点群①と 3D 都市モデルから出力した点群②をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）を出力する。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに公知の論文情報等を参照しつつ、Pretia の点群出力アルゴリズムで抽出した①及び②の点群の対応に最適化するよう調整を行う
自己位置推定機能	VPS	FN507	PLATEAU-to-Pretia VPS（自己位置推定アルゴリズム）	● 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）を返す処理を行う。本プロジェクトでは、Pretia が所有するアルゴリズムをそのまま適用す

				る
	座標変換	FN508	座標変換アルゴリズム	● PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換する処理。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに一部調整を行う
エンドユーザーアプリ		FN509	自己位置推定用画像撮影	● ユーザーアプリケーション上で、自己位置推定処理に必要な画像を撮影し、Pretia サーバへ送信する
		FN510	コンテンツ表示	● Pretia サーバにて演算した自己位置情報をもとに、3D 都市モデルなどのデジタルコンテンツを AR 表示する
Unity		FN511	コンテンツ位置指定	● Unity 開発画面に取り込んだ PLATEAU の 3D 都市モデル上に表示させる 3D コンテンツを配置し、コンテンツ表示位置を指定する
		FN512	AR コンテンツビルド	● スマートフォンアプリとして利用するため、開発したアプリをビルドする

7-2-2. 利用したソフトウェア・ライブラリ

表 7-3 利用したソフトウェア・ライブラリ

※赤文字：既存改修・新規開発

ID	項目	内容
SL501	Pretia 点群出力	● 複数枚の画像から対象の形状を復元する SfM(Structure from motion)をベースとした Pretia が既存で所有する独自ソフトウェア。画像内の特徴点をもとに VPS 用の点群を出力する。（3D 都市モデルからの点群出力アルゴリズムの要素技術として活用する）
SL502	点群 to 点群対応化	● 異なる 2 つの点群データ（①現実世界から変換した点群データと②3D 都市モデルから変換した点群データ）をもとに座標情報等を整理し双方の対応関係を導出し、同一の処理で扱える点群へ変換するソフトウェア ● 本プロジェクトでは、Pretia が既存で保有するアルゴリズムをベースに公知の論文情報等を参照しながらモデル及びパラメータを調整し、Pretia の点群出力アルゴリズムで抽出した①及び②の点群の対応に最適化する
SL503	Unity	● Unity 社が提供するゲーム開発プラットフォーム
SL504	Pretia サーバ	● Pretia が提供する AR 開発プラットフォーム「Pretia」のデータ処理サーバ。本プロジェクトでは、Pretia サーバの持つ VPS 用点群出力処理を利用する
SL505	PretiaSDK	● Pretia が提供する AR 開発プラットフォーム「Pretia」の SDK。Pretia サーバと連携し Unity 上で AR コンテンツを開発するためのソフトウェア開発キット
SL506	PLATEAU to PretiaVPS	● Pretia が提供する AR 開発プラットフォーム「Pretia」などで独自開発したアルゴリズムを含む既存のソフトウェア。本プロジェクトではこの Pretia 所有のアルゴリズムに対して、モデル及びパラメータなどの調整を行うことで、3D 都市モデルからの点群出力及び点群を利用した自己位置推定などの機能を研究開発する
SL507	座標変換	● PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換するソフトウェア。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに一部調整を行う
SL508	PLATEAU SDK for Unity	● PLATEAU から提供される、GityGML を各種データ形式に変

		換し Unity へ取り込むソフトウェア

7-2-3. 開発機能の詳細要件

開発機能の詳細要件を記す。なお、本業務において新規開発した要素（機能名）を赤字で示す。

1)　機能一覧

1.　【FN501】点群出力用画像撮影

- 機能概要
  - 専用のレコーダーで現実世界のカメラ画像撮影・Pretia サーバへ送信する。

- フローチャート

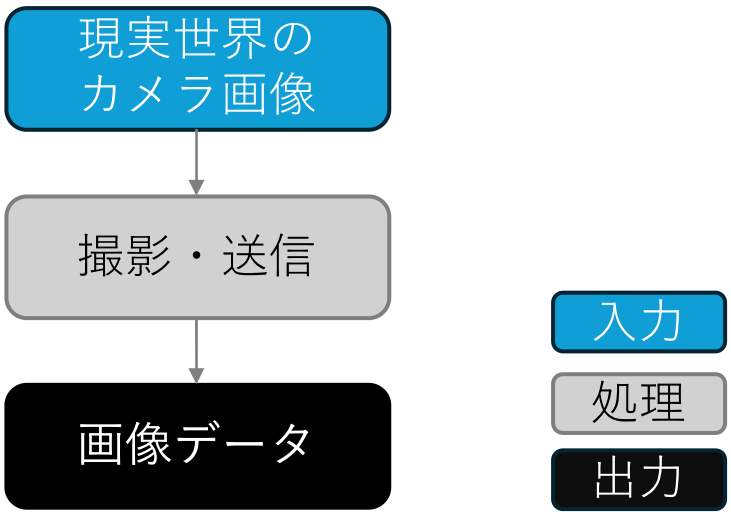


図 7-7 点群出力用画像撮影処理のフローチャート

- データ仕様
  - 入力
    - スマートフォンカメラ画像
      - 内容
        - 専用のレコーダーで撮影した現地のカメラ画像データ
      - 形式
        - バイナリ形式
      - データ詳細
        - 入力インターフェース【IF501】を参照
  - 出力
    - 画像データ
      - 内容
        - 専用のレコーダーで撮影した現地のカメラ画像データ
      - 形式

- PNG 形式
- データ詳細
  - 出力インターフェース【IF601】を参照

- 機能詳細
  - 点群出力用画像撮影
    - 処理内容
      - 専用のレコーダーで現実世界のカメラ画像データを取得し Pretia サーバへ送信する。
    - 利用するライブラリ
      - なし
    - 利用するアルゴリズム
      - なし

2. 【FN502】VPS 用点群出力処理

- 機能概要
  - 専用のレコーダーで取得した現実世界のカメラ画像データを、SfM をベースとした VPS 用点群出力処理を用いて点群データを出力する。
- フローチャート

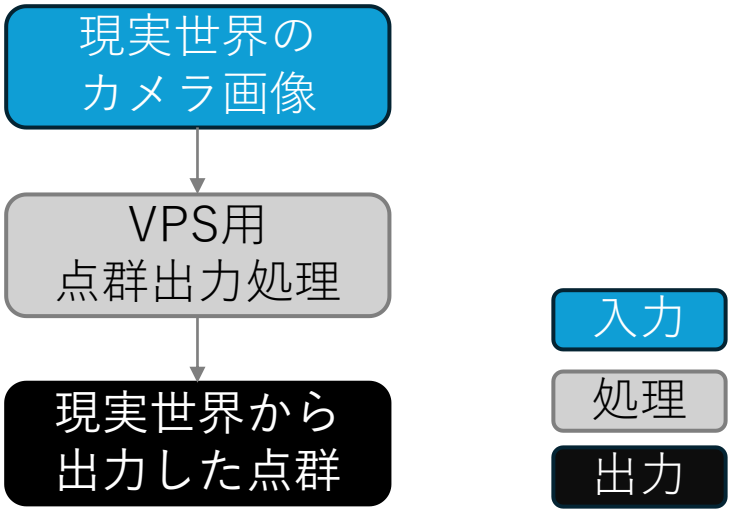


図 7-7 VPS 用点群出力処理のフローチャート

- データ仕様
  - 入力
    - 現実世界のカメラ画像
      - 内容
        - 専用のレコーダーで撮影した現地のカメラ画像データ
      - 形式
        - PNG 形式

- データ詳細
  - 入力インターフェース【IF505】を参照

- 出力
  - 現実世界から出力した点群
    - 内容
      - 専用のレコーダーで撮影した現地のカメラ画像データをもとに出力した 3D の点群データ
    - 形式
      - json 形式
    - データ詳細
      - 内部連携インターフェース【IF703】を参照

- 機能詳細
  - VPS 用点群出力
    - 処理内容
      - 専用のレコーダーで取得した現実世界のカメラ画像データを SfM をベースとした VPS 用点群出力処理を用いて点群データを出力する。
    - 利用するライブラリ
      - Pretia 点群出力（ソフトウェア・ライブラリ【SL501】を参照）
    - 利用するアルゴリズム
      - 点群出力アルゴリズム（アルゴリズム【AL501】を参照）

3. 【FN503】CityGML の読込・変換

- 機能概要
  - PLATEAU SDK により CityGML を OBJ ファイルへ変換し Unity 上の開発環境へ取り込む。
- フローチャート

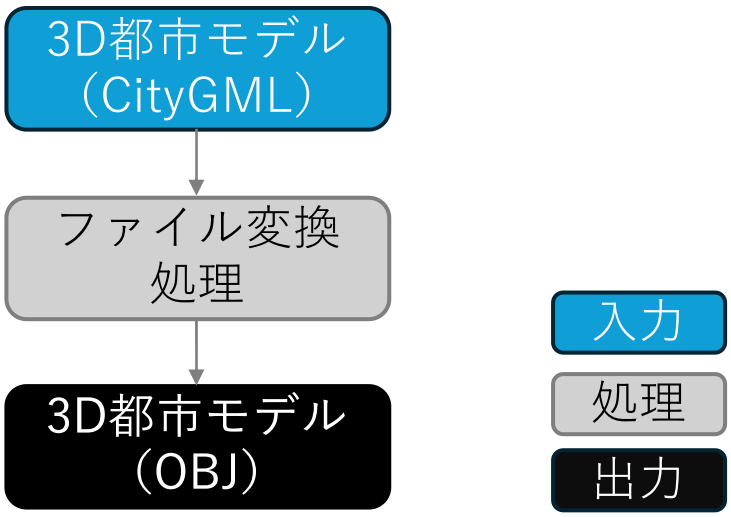


図 7-7 CityGML の読込・変換処理のフローチャート

- データ仕様
  - 入力
    - CityGML ファイル
      - 内容
        - 3D 都市モデルの CityGML データ
      - 形式
        - CityGML 形式
      - データ詳細
        - 入力インターフェース【IF502】を参照
  - 出力
    - OBJ ファイル
      - 内容
        - PLATEAU SDK にて変換した OBJ 形式の 3D 都市モデル
      - 形式
        - OBJ 形式
      - データ詳細
        - 内部連携インターフェース【IF701】を参照

- 機能詳細
  - CityGML の読込・変換
    - 処理内容

- PLATEAU SDK により CityGML 形式の 3D 都市モデルを読込み・OBJ 形式へ変換、Unity 上へ取り込む

✧ 利用するライブラリ

- PLAETAU SDK for Unity（ソフトウェア・ライブラリ【SL508】を参照）

✧ 利用するアルゴリズム

- なし

4. 【FN504】3D 都市モデルの点群出力用画像撮影

- 機能概要
  - ➢ 点群出力の前処理として、Unity 上に取り込んだ 3D 都市モデルをシミュレーションカメラで撮影し PNG ファイルへ変換する。
- フローチャート

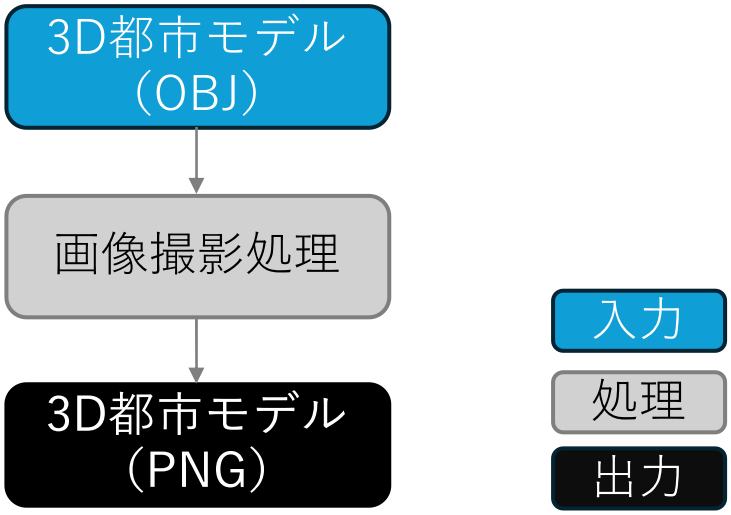


図 7-7 3D 都市モデルの点群出力用画像撮影処理のフローチャート

- データ仕様
  - ➢ 入力
    - ✧ OBJ ファイル形式の 3D 都市モデル
      - 内容
        - ➢ Unity 上に取り込んだ 3D 都市モデル
      - 形式
        - ➢ OBJ 形式
      - データ詳細
        - ➢ 内部連携インターフェース【IF701】を参照
  - ➢ 出力
    - ✧ 3D 都市モデルのスナップショット画像データ
      - 内容
        - ➢ Unity シミュレーションカメラで撮影した 3D 都市モデルの画像データ

- 形式
  - PNG 形式
- データ詳細
  - 出力インターフェース【IF602】を参照

- 機能詳細
  - 3D 都市モデルの点群出力用画像撮影
    - 処理内容
      - Unity のシミュレーションカメラにより、3D 都市モデルを画像データへ変換する。数千枚の画像が必要なため、変換にあたっては簡易ツールを作成し、半自動で出力した。
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL503】を参照）
    - 利用するアルゴリズム
      - なし

5. 【FN505】3D 都市モデルからの点群出力処理

- 機能概要
  - 3D 都市モデルの画像データを、SfM をベースとした VPS 用点群出力処理を用いて点群データを出力する。
- フローチャート

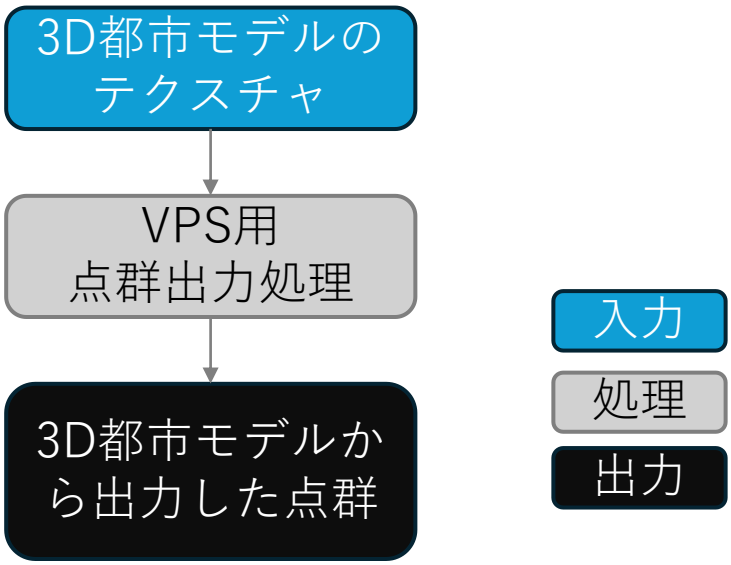


図 7-8 3D 都市モデルからの点群出力処理のフローチャート

- データ仕様
  - 入力
    - 3D 都市モデルの画像データ
      - 内容

- Unity 上に取り込んだ 3D 都市モデルをもとに、Unity カメラを利用し撮影したシミュレーション画像
- 形式
  - PNG 形式
- データ詳細
  - 入力インターフェース【IF506】を参照

- 出力
  - 3D 都市モデルから出力した点群
    - 内容
      - Unity 上で取得した 3D 都市モデルの画像データをもとに出力した 3D の点群データ
    - 形式
      - json 形式
    - データ詳細
      - 内部連携インターフェース【IF704】を参照

- 機能詳細
  - 3D 都市モデルからの点群出力
    - 処理内容
      - Unity 上で取得した 3D 都市モデルの画像データを、SfM をベースとした VPS 用点群出力処理を用いて点群データを出力する。
    - 利用するライブラリ
      - Pretia 点群出力（ソフトウェア・ライブラリ【SL501】を参照）
    - 利用するアルゴリズム
      - 点群出力アルゴリズム（アルゴリズム【AL501】を参照）

6. 【FN506】点群 to 点群対応化アルゴリズム

- 機能概要
  - 現実世界から出力した点群①と 3D 都市モデルから出力した点群②をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）を出力する。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに公知の論文情報等を参照しつつ、Pretia の点群出力アルゴリズムで抽出した①及び②の点群の対応に最適化するよう調整を行う。
  - Pretia VPS は画像情報を元にベクトル情報を含む空間マップ（点群）を作成し、２時点でのマッチングにより自己位置推定を行う。そのため、空間マップ（点群）と自己位置推定時のカメラから取得した点群に大きな差分がある場合、適切に自己位置推定を実施できない。3D 都市モデル（LOD3）は建物の形状や縮尺などは正確にとらえているものの、外壁のテクスチャは現実と異なる部分も多いほか、現実世界の環境は日々変化するため、単純に 3D 都市モデルを空間マップとして使用することはできない。この課題を解決するため、本点群 to 点群対応化アルゴリズムによる２つの点群に対する対応関係の導出が重要となる。
- フローチャート

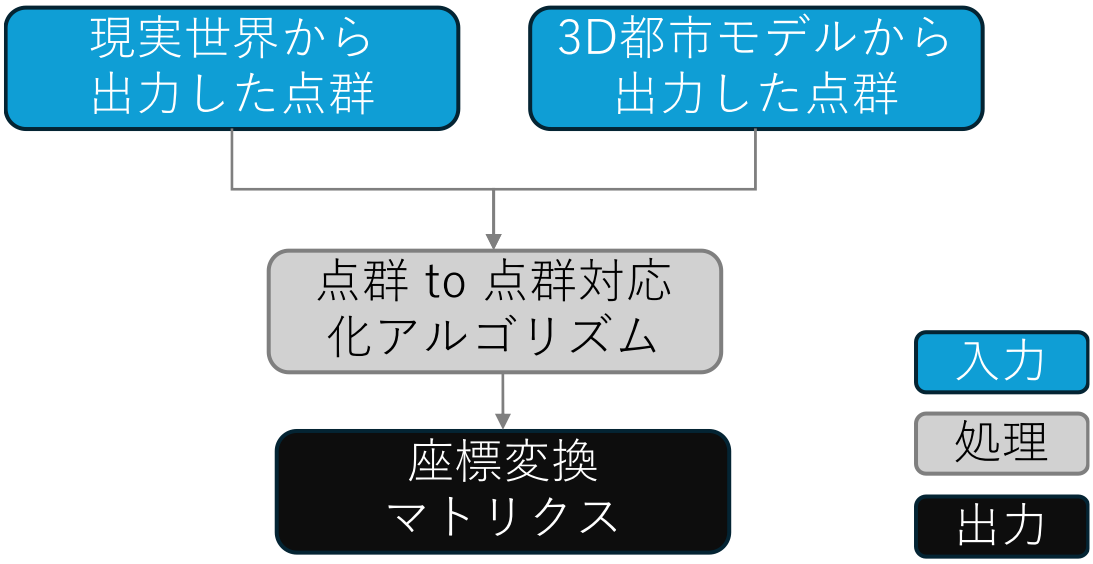


図 7-9 点群 to 点群対応化アルゴリズムのフローチャート

- データ仕様
  - 入力
    - 現実世界から出力した点群
      - 内容
        - 専用のレコーダーで撮影した現地のカメラ画像データをもとに出力した 3D の点群データ
      - 形式
        - json 形式
      - データ詳細

- 内部連携インターフェース【IF703】を参照
- 3D 都市モデルから出力した点群
  - 内容
    - Unity 上で取得した 3D 都市モデルの画像データをもとに出力した 3D の点群データ
  - 形式
    - json 形式
  - データ詳細
    - 内部連携インターフェース【IF704】を参照
- 出力
  - 座標変換マトリクス
    - 内容
      - 現実世界から出力した点群と 3D 都市モデルから出力した点群をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）
    - 形式
      - json 形式
    - データ詳細
      - 内部連携インターフェース【IF705】を参照

- 機能詳細
  - 点群 to 点群対応化
    - 処理内容
      - 現実世界から出力した点群と 3D 都市モデルから出力した点群をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）を出力する。
    - 利用するライブラリ
      - 点群 to 点群対応化（ソフトウェア・ライブラリ【SL502】を参照）
    - 利用するアルゴリズム
      - 点群 to 点群対応化アルゴリズム（アルゴリズム【AL601】を参照）

7. 【FN507】PLATEAU-to-Pretia VPS（自己位置推定アルゴリズム）

- 機能概要
  - 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）を返す処理をおこなう。本プロジェクトでは、Pretia が所有するアルゴリズムをそのまま適用する。
- フローチャート

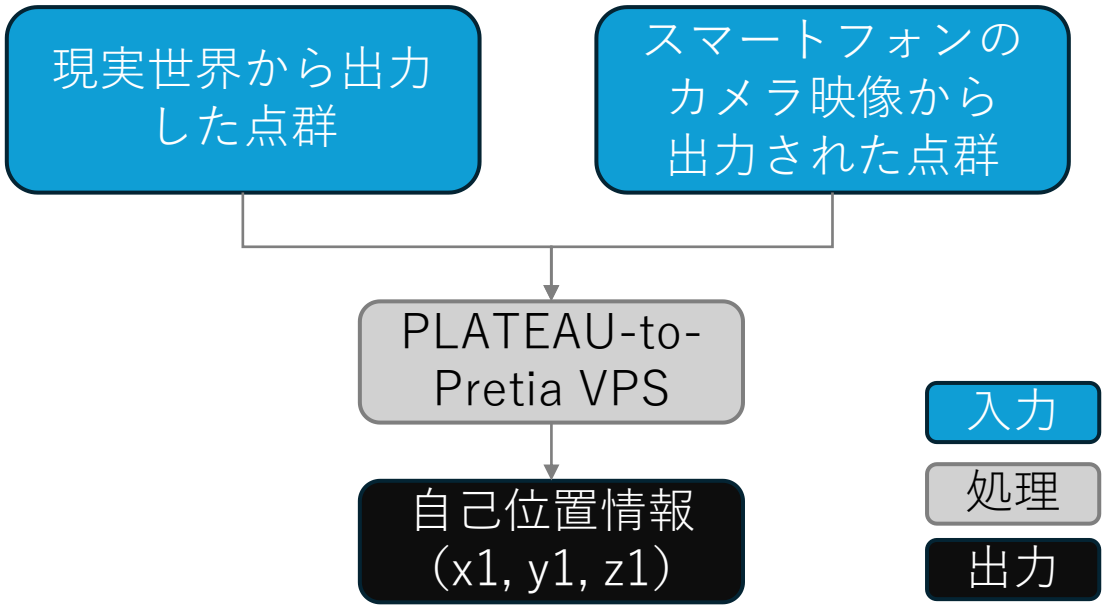


図 7-10 PLATEAU-to-Pretia VPS（自己位置推定アルゴリズム）のフローチャート

- データ仕様
  - 入力
    - 現実世界から出力した点群
      - 内容
        - 専用のレコーダーで撮影した現地のカメラ画像データをもとに出力した 3D の点群データ
      - 形式
        - json 形式
      - データ詳細
        - 内部連携インターフェース【IF703】を参照
    - スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群
      - 内容
        - スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群データ
      - 形式
        - json 形式
      - データ詳細

- 入力インターフェース【IF508】を参照
- 出力
  - 自己位置情報（x1, y1, z1）
    - 内容
      - 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）として算出したデータ
    - 形式
      - json 形式
    - データ詳細
      - 内部連携インターフェース【IF706】を参照

- 機能詳細
  - 自己位置推定処理
    - 処理内容
      - 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）を出力する。
    - 利用するライブラリ
      - PLATEAU to Pretia VPS（ソフトウェア・ライブラリ【SL506】を参照）
    - 利用するアルゴリズム
      - 自己位置推定アルゴリズム（アルゴリズム【AL502】を参照）

8. 【FN508】座標変換アルゴリズム

- 機能概要
  - PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換する処理。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに一部調整を行う。
- フローチャート

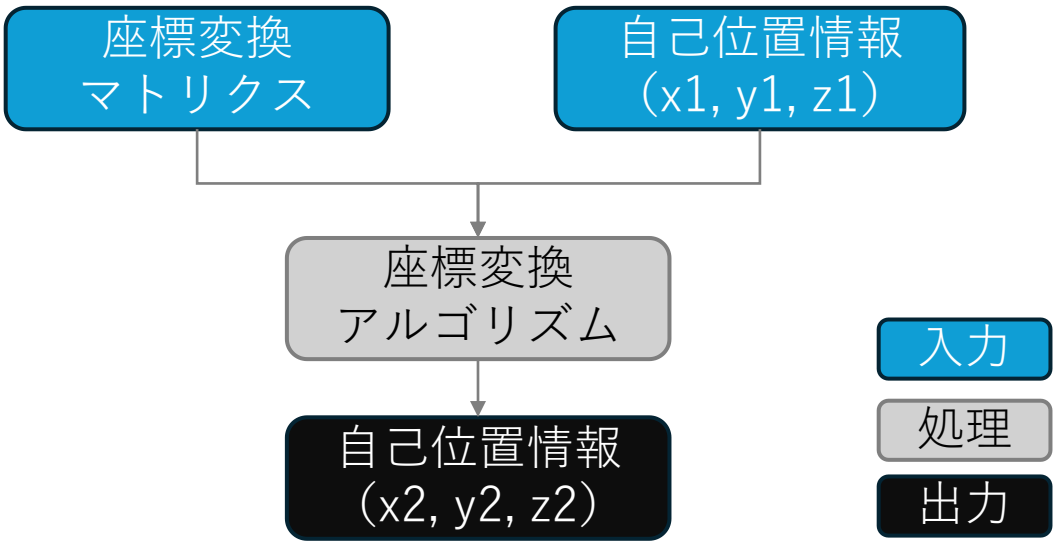


図 7-11 座標変換アルゴリズムのフローチャート

- データ仕様
  - 入力
    - 座標変換マトリクス
      - 内容
        - 現実世界から出力した点群と 3D 都市モデルから出力した点群をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）
      - 形式
        - json 形式
      - データ詳細
        - 内部連携インターフェース【IF705】を参照
    - 自己位置情報
      - 内容
        - 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし作成した自己位置情報（x1, y1, z1）。
      - 形式

        - json 形式
    - データ詳細
        - 内部連携インターフェース【IF706】を参照
- 出力
    - 3D 都市モデルに最適化した自己位置情報
        - 内容
            - PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で変換し取得した 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）
        - 形式
            - json 形式
        - データ詳細
            - 出力インターフェース【IF605】を参照
- 機能詳細
    - 座標変換アルゴリズム
        - 処理内容
            - PLATEAU-to-Pretia VPS において出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換する処理。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに一部調整を行う。
        - 利用するライブラリ
            - 座標変換（ソフトウェア・ライブラリ【SL507】を参照）
        - 利用するアルゴリズム
            - 座標変換アルゴリズム（アルゴリズム【AL602】を参照）

9. 【FN509】自己位置推定用画像撮影

- 機能概要
  - ユーザーアプリケーション上で自己位置推定に必要な画像を撮影し Pretia サーバへ送信する。
- フローチャート

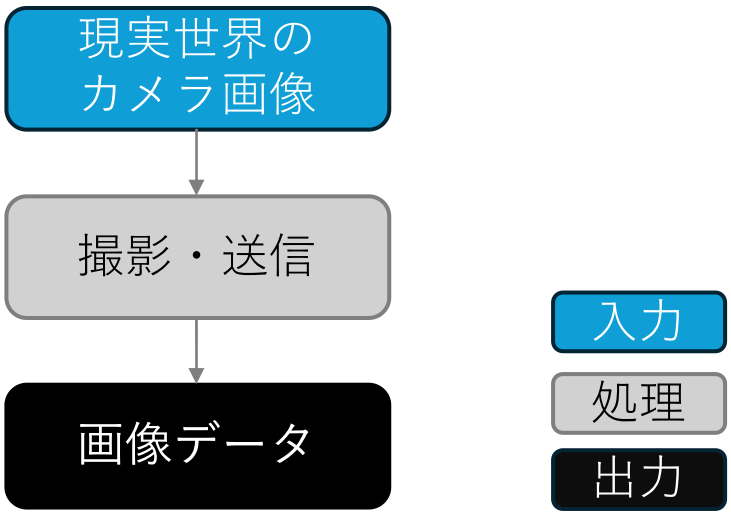


図 7-7 自己位置推定用画像撮影処理のフローチャート

- データ仕様
  - 入力
    - スマートフォンのカメラ画像
      - 内容
        - ユーザーアプリケーション上で撮影したカメラ画像
      - 形式
        - バイナリ形式
      - データ詳細
        - 入力インターフェース【IF504】を参照
  - 出力
    - 画像データ
      - 内容
        - ユーザーアプリケーション上で撮影したカメラ画像
      - 形式
        - PNG 形式
      - データ詳細
        - 出力インターフェース【IF604】を参照

- 機能詳細
  - 自己位置推定用画像撮影
    - 処理内容

- ユーザーアプリケーション上で自己位置推定に必要な画像を撮影し Pretia サーバへ送信する。

✧ 利用するライブラリ

- なし

✧ 利用するアルゴリズム

- なし

10. 【FN510】コンテンツ表示

- 機能概要
  - Pretia サーバにて演算した自己位置情報をもとに、3D 都市モデルなどのデジタルコンテンツを AR 表示する
- フローチャート

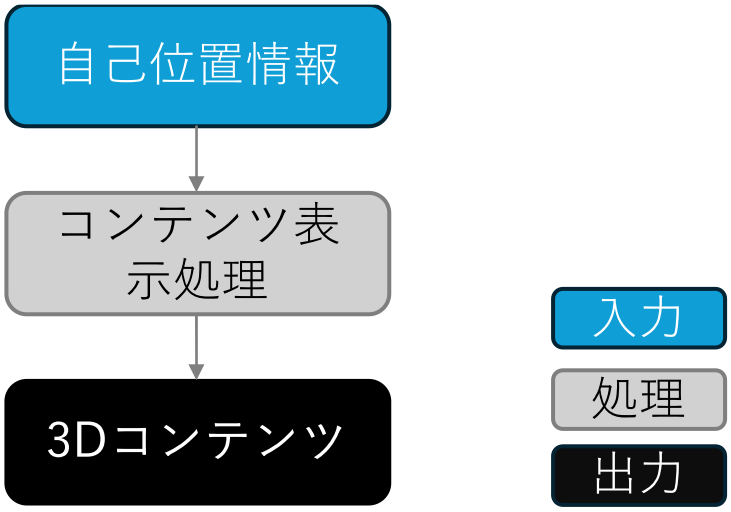


図 7-7 コンテンツ表示処理のフローチャート

- データ仕様
  - 入力
    - ✧ 自己位置情報
      - 内容
        - Pretia サーバにて演算した自己位置情報（x, y, z）
      - 形式
        - バイナリデータ
      - データ詳細
        - 入力インターフェース【IF507】を参照
  - 出力
    - ✧ 3D コンテンツ
      - 内容

- Unity 上で事前に表示位置を指定した 3D コンテンツ
- 形式
  - OBJ 形式
- データ詳細
  - 出力インターフェース【IF606】を参照

- 機能詳細
  - コンテンツ表示
    - 処理内容
      - Pretia サーバにて演算した自己位置情報をもとに、3D 都市モデルなどのデジタルコンテンツをユーザーアプリ上に AR 表示する
    - 利用するライブラリ
      - なし
    - 利用するアルゴリズム
      - なし

11. 【FN511】コンテンツ位置指定

- 機能概要
  - ユーザーアプリ上で AR 表示する 3D コンテンツの表示位置を指定する。
- フローチャート

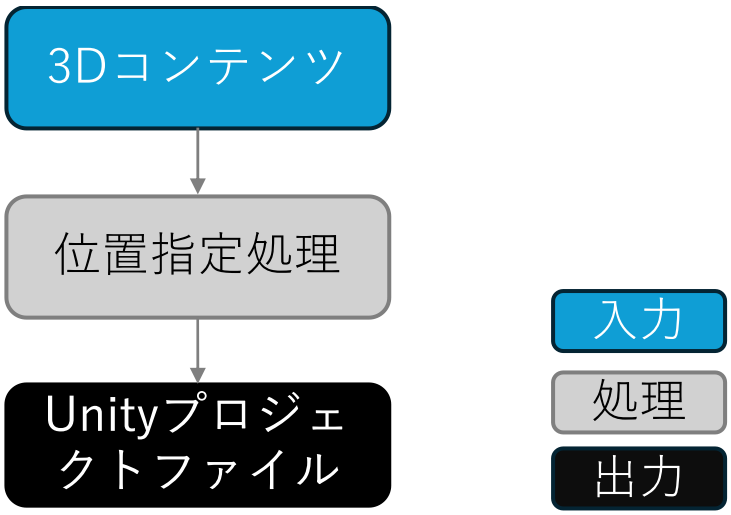


図 7-7 コンテンツ位置指定処理のフローチャート

- データ仕様
  - 入力
    - 3D コンテンツ
      - 内容

- ユーザーアプリ上で AR 表示する 3D コンテンツ
- 形式
  - OBJ 形式
- データ詳細
  - 入力インターフェース【IF503】を参照

- 出力
  - Unity プロジェクトファイル
    - 内容
      - 管理情報（yaml ファイル）とコンテンツ(OBJ, PNG ファイル)をまとめたディレクトリー式
    - 形式
      - yaml 形式, OBJ 形式, PNG 形式
    - データ詳細
      - 内部連携インターフェース【IF702】を参照

- 機能詳細
  - コンテンツ位置指定
    - 処理内容
      - ユーザーアプリ上で AR 表示する 3D コンテンツの表示位置を指定する。
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL503】を参照）
    - 利用するアルゴリズム
      - なし

12. 【FN512】AR コンテンツビルド

- 機能概要
  - 開発したコンテンツをスマートフォンアプリとして利用するため、ビルドする。
- フローチャート

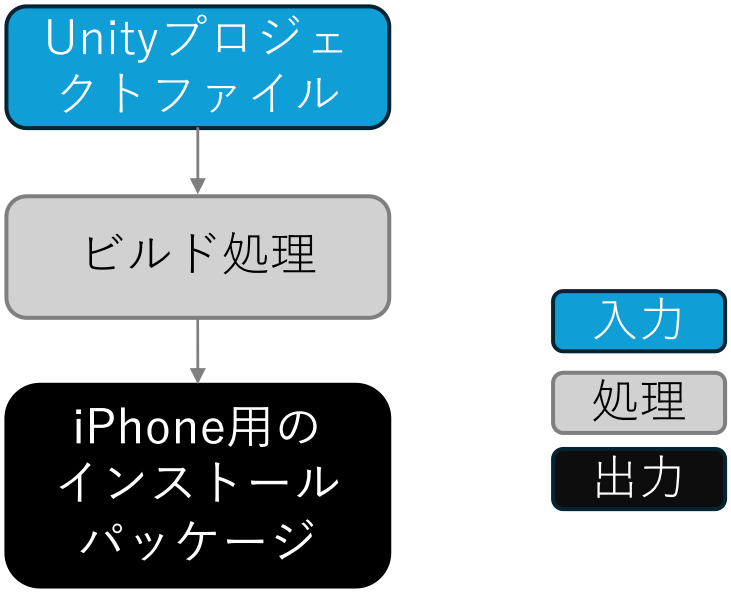


図 7-7 AR コンテンツビルド処理のフローチャート

- データ仕様
  - 入力
    - Unity プロジェクトファイル
      - 内容
        - 管理情報（yaml ファイル）とコンテンツ(OBJ, PNG ファイル)をまとめたディレクトリー式
      - 形式
        - yaml 形式, OBJ 形式, PNG 形式
      - データ詳細
        - 内部連携インターフェース【IF702】を参照
  - 出力
    - iPhone 用のインストールパッケージ
      - 内容
        - Unity 上でビルドした iPhone 用のアプリインストールパッケージ
      - 形式
        - ipa 形式
      - データ詳細
        - 出力インターフェース【IF603】を参照

- 機能詳細
  - AR コンテンツビルド
    - 処理内容
      - 開発したコンテンツをスマートフォンアプリとして利用するため、ビルドする。
    - 利用するライブラリ
      - Unity（ソフトウェア・ライブラリ【SL503】を参照）
    - 利用するアルゴリズム
      - なし

7-3. アルゴリズム

7-3-1. 利用したアルゴリズム

今回の実証実験では以下のアルゴリズムを利用した。

表 7-4 利用したアルゴリズム一覧

ID	アルゴリズムを利用した機能	名称	説明	選定理由
AL501	FN502	点群出力アルゴリズム	● 画像ファイルを入力とし、VPS 用の点群を出力するアルゴリズム。Pretia が所有するアルゴリズムをそのまま適用する。	● Pretia が所有するアルゴリズムであり詳細なパラメータ調整が可能なため。 ● LiDAR などの高価なセンサが不要なため
AL502	FN504	自己位置推定アルゴリズム（PLATEAU-to-Pretia VPS）	● 実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）を返す処理を行う。本プロジェクトでは、Pretia が所有するアルゴリズムをそのまま適用する。	● Pretia が所有するアルゴリズムであり詳細なパラメータ調整が可能なため。 ● LiDAR などの高価なセンサが不要なため

1. 【AL501】点群出力アルゴリズム

今回の実証実験における点群出力アルゴリズムは Pretia が所有するアルゴリズム、具体的には SfM（Structure from Motion）を応用したアルゴリズムを利用する。利用するアルゴリズムの基本的な処理の流れは下記図 7-7 のとおり。

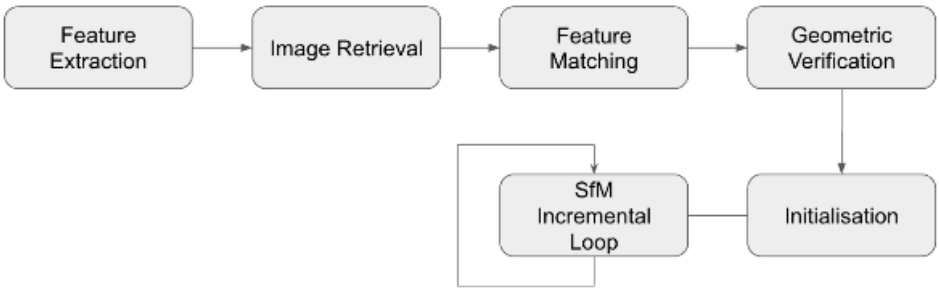


図 7-7 SfM（Structure from Motion）の概略処理フロー

- Feature Extraction：画像上から特徴点を抽出する処理
- Image Retrieval：関係性の高そうな画像の絞り込みを行う処理
- Feature Matching：特徴点のマッチングを行う処理
- Geometric Verification：マッチングした特徴点の妥当性を確認する処理
- Initialisation：SfM により 3D の点群マップを作成するに当たっての初期化処理。
- SfM Incremental Loop：画像から 3D の点群マップを構築するためのメイン処理。SfM のメイン処理は以下の 5 つのような技術をもとにループ構造で構築される。

1. Motion Estimation
   連続する画像間でのカメラまたはオブジェクトの動きを推定する。画像間の相対的な位置関係を理解するのに利用される。

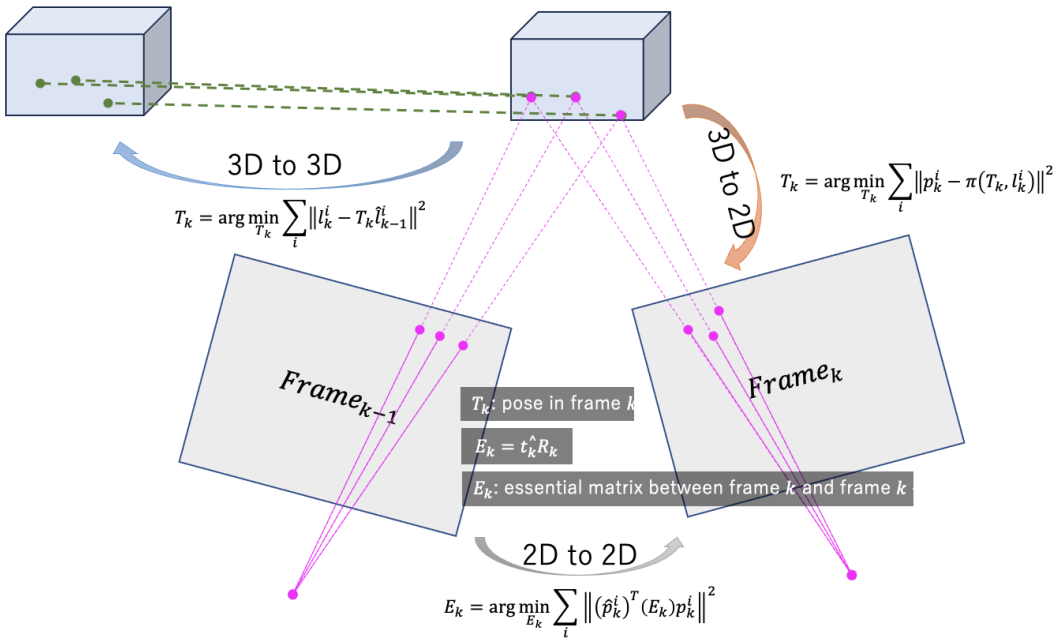


図 7-8 Motion Estimatio の処理イメージ

2. Image Registration (PnP)
   Perspective-n-Point (PnP) アルゴリズムを使用して、画像間の特徴点をもとにカメラの位置と向きを推定する。これにより、各画像が撮影された際のカメラの 3D の姿勢が求められる。
3. Triangulation
   マッチングされた特徴点から 3D 空間内のポイントの位置を三角測量によって決定する。
4. Bundle Adjustment
   カメラの位置、向き、及び 3D 空間内のポイントの位置を同時に微調整し、再投影誤差（再度 2D 画像に投影した際の、予測された画像上の位置と実際の観測位置との差）を最小限に抑えることで、より正確な 3D の点群を構築する。
5. Next Best View Selection
   どの画像が次に最適な 3D 点群情報を提供するかを決定する。例えば、5 枚の画像 A, B, C, D, E があり、最初に A と B を使用して 3D 再構築した場合、Next Best View Selection のプロセスでは、残りの画像 C, D, E 内に A と B から再構築された 3D 点群と共通の特徴を持つピクセルを識別する。これらの共通特徴を持つピクセルの数や分布に基づいて、どの画像が次に最適な視点を提供するかをスコアリングすることで、どの画像が次に最適な 3D 情報を提供するかを決定する。

2. 【AL502】自己位置推定アルゴリズム（PLATEAU-to-Pretia VPS）

自己位置推定アルゴリズムでは、【AL501】で実世界から出力した点群に対して、スマートフォン向け AR アプリケーションから新たに取得した画像をもとに出力された点群をマッチングし、自己位置情報(x1, y1, z1)を特定するアルゴリズムである。具体的には、下記のようなプロセスを実施する。各プロセスで実施する処理は、【AL501】と同様となる。

1. Feature Extraction
2. Feature Matching
3. Image Registration (PnP)
4. Triangulation
5. Bundle Adjustment

7-3-2. 開発したアルゴリズム

7-3-2-a. 開発したアルゴリズムの概要

今回の実証実験では以下 2 つのアルゴリズムを開発した。

NO.	名称	概要	利用目的
AL601	点群 to 点群対応化アルゴリズム	● 異なる 2 つの点群データをもとに座標情報等を整理し双方の対応関係を導出し、同一の処理で扱える点群へ変換するアルゴリズム ● 本プロジェクトでは、Pretia が既存で保有するアルゴリズムをベースに公知の論文情報等を参照しながらモデル及びパラメータを調整し、Pretia の点群出力アルゴリズムで抽出した①及び②の点群の対応に最適化する	①現実世界から変換した点群データと②3D 都市モデルから変換した点群データの対応関係を導出し、座標変換マトリクスを出力する
AL602	座標変換アルゴリズム	● PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換する処理。本プロジェクトでは、 ● Pretia が所有するアルゴリズムをベースに一部調整を行う	現実世界から変換した点群データの座標系から 3D 都市モデルから変換した点群データの座標系へ変換する

7-3-2-b. 開発したアルゴリズムの詳細

1) 【AL601】点群 to 点群対応化アルゴリズム

- 本アルゴリズムを利用した機能
  - 【FN506】

- アルゴリズムの詳細

①現実世界から出力した点群と②3D 都市モデルから出力した点群をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）を出力する。

Pretia VPS は画像情報を元にベクトル情報を含む空間マップ（点群）を作成し、2 時点でのマッチングにより自己位置推定を行う。そのため、空間マップ（点群）と自己位置推定時のカメラから取得した点群に大きな差分がある場合、適切に自己位置推定を実施できない。3D 都市モデル（LOD3）は建物の形状や縮尺などは正確にとらえているものの、外壁のテクスチャは現実と異なる部分も多いほか、現実世界の環境は日々変化するため、単純に 3D 都市モデルを空間マップとして使用することはできない。この課題を解決し PretiaVPS をベースに 3D 都市モデルを活用した自己位置推定を実現するため、本点群 to 点群対応化アルゴリズムによる 2 つの点群に対する対応関係の導出が重要となる。

点群 to 点群の対応化については、点群出力のアルゴリズムなどの組合せに、いくつかの論文で成果を出している。本プロジェクトでは、Pretia が所有する点群 to 点群のアルゴリズムをベースに公知の論文情報等を参照しつつ、Pretia の点群出力アルゴリズムで抽出した①及び②の点群の対応に最適化するよう調整を行う。本アルゴリズムは本プロジェクトにおけるコア技術となる。
本プロジェクトでは、本章下部に記載の 6 つの文献をもとに最適な点群 to 点群のアルゴリズムを参考に検討し下記 2 つのアルゴリズムを開発した。

【アルゴリズム 1】

本章下部の 5 番目に記載した論文を参考に機械学習によるアプローチを取り入れたアルゴリズムを開発した。処理フローには 2 つの学習モデルを用いる。一つは点群を元に特徴ベクトルへ変換するモデル（学習モデル A）、もう一つは 2 つの特徴ベクトルデータを元に、座標変換マトリクスを出力するためのモデル（学習モデル B）である。それぞれの学習モデル及び適用する学習モデル用のパラメータは公開されているオープンデータ(3D Cross-source point cloud registration (3DCSR) benchmark [11])を活用し作成する。

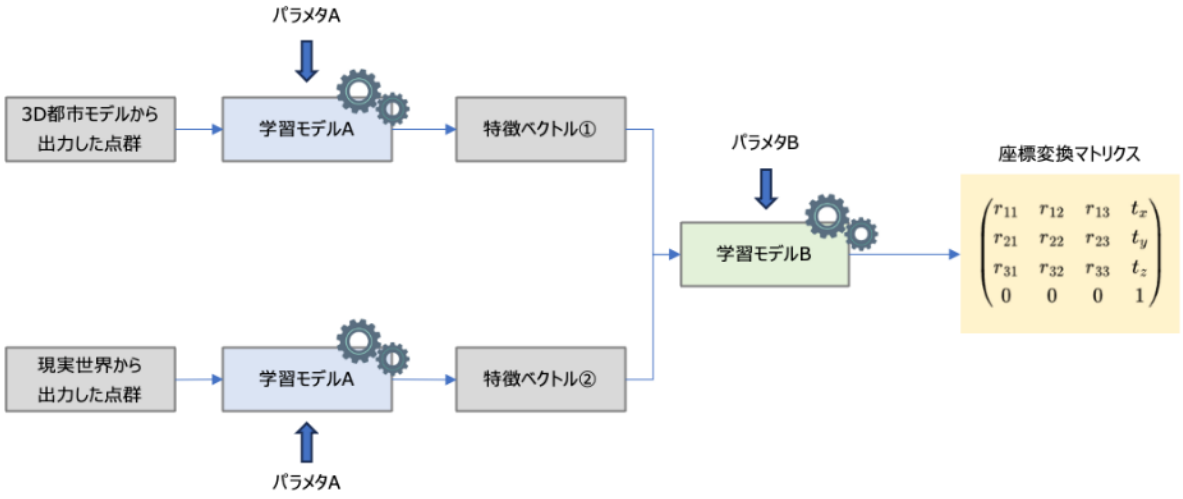


図 7-9 機械学習を用いた点群 to 点群対応化アルゴリズムの処理フロー

[11] Univesity of Technology Sydney.“3D Cross-source point cloud registration (3DCSR) benchmark”. https://multimediauts.org/3D_data_for_registration, (2024/2/6)

学習モデル A のトレーニング

- 学習モデル A のトレーニングには、 3 種類の点群（ベースとなる点群、 対応化すべきソースの異なる点群、 対応化できないソースの点群）を同一の学習モデル A に読み込ませることで実施する。この時、 3 つの点群から学習モデル A を介して出力した 3 つの特徴ベクトルを元に、損失関数によりこの損失が最小となるような学習モデル A のパラメータを最適化する。

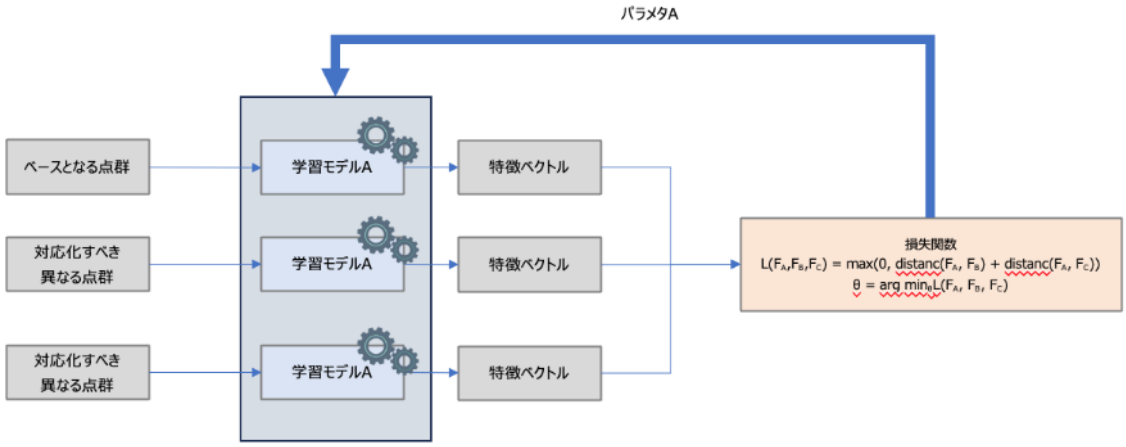


図 7-10 学習モデル A のトレーニングフロー

学習モデル B のトレーニング

- 学習モデル B のトレーニングには、 2 種類の点群（ベースとなる点群、 対応化すべきソースの異なる点群）をもとに学習モデル A で出力した 2 つの特徴ベクトルを元に学習モデル B に読み込ませることで座標変換マトリクスを取得。あらかじめオープンデータとして入手している「正しい座標変換マトリクスとともに損失関数へインプットすることで、学習モデル B のパラメータを最適化する。

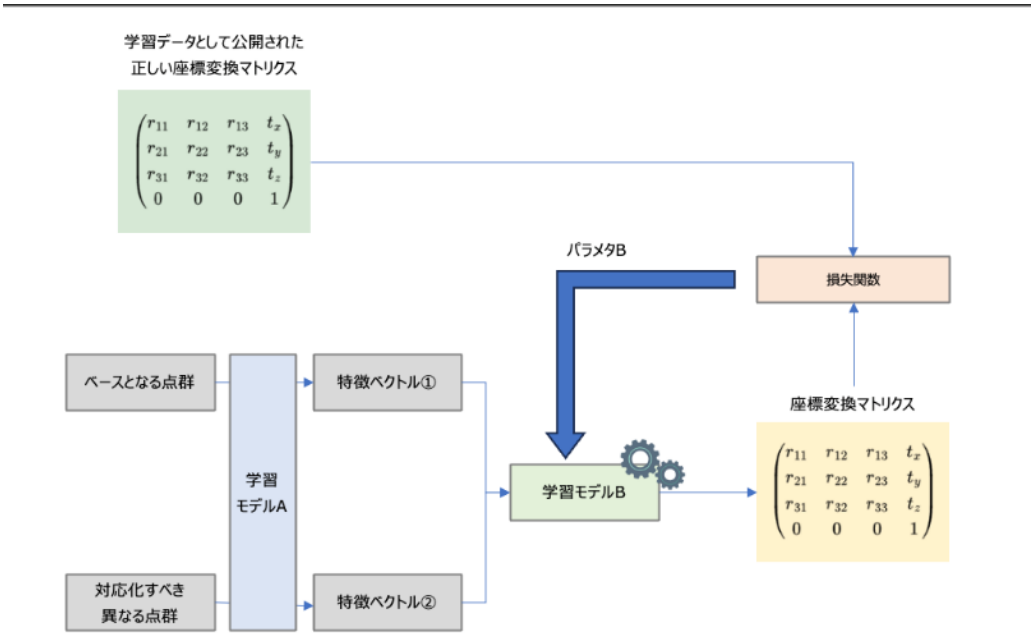


図 7-11 学習モデル B のトレーニングフロー

【アルゴリズム 2】

No1-3 のアプローチを参考に、機械学習を使わないデータ処理による対応化アルゴリズムを開発した処理フローは Step1 と Step2 の 2 つのセクションに分けて実行する。Step1 では 3D 都市モデル及び現実世界から取得した点群を一定のグリッドで切り分ける。Step2 では Step1 できり分けた小さな領域ごとに ICP（iterative closest point）アルゴリズムで対応化を実施する。ICP アルゴリズムは異なるデータセット間の三次元点群に対して位置合わせを行う際に用いるアルゴリズムであり、基準となるデータが持つ点群テータに対して入力された点群データを重ね合わせ、2 つの点群間の距離が最小となるためにはどの点同士を対応させるべきかということを判断し、マッチングすることが出来る。

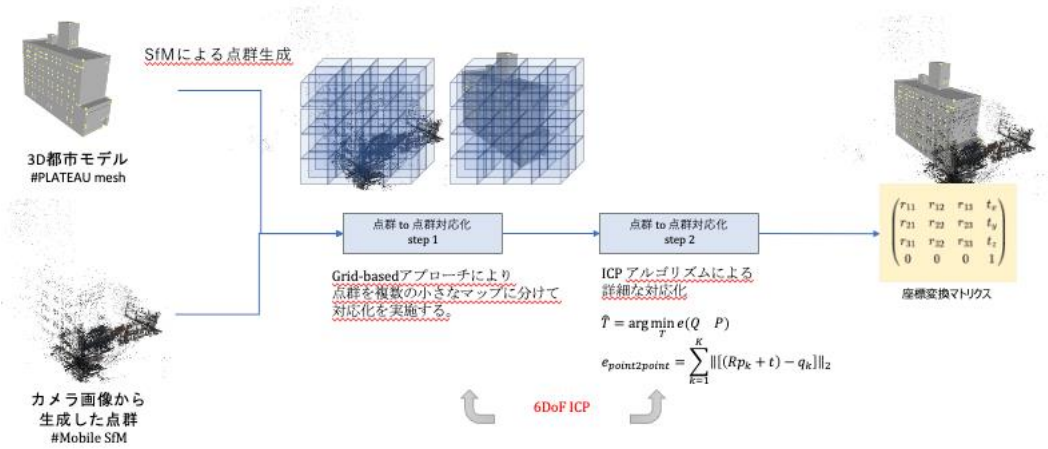


図 7-12 アルゴリズム 2 における点群 to 点群対応化

- 参考とした 6 つの論文

各論文の抽出に際しては、下記 2 つの観点で調査を行っている。

① 点群の対応付けに際して「各点ごとの対応付けを実施する」手法と「半教師あり学習による対応付けを実施する」手法を比較評価すること。
② 対応付けする点群の出力アルゴリズムが本プロジェクトと近しいこと。（画像をもとに SfM 等のアルゴリズムで抽出した点群を利用している）

具体的に参考にした参考を以下に記載する。

1. *“Street view cross-sourced point cloud matching and registration. Furong Peng, Qiang Wu, Lixin Fan, Jian Zhang, Yu You, Jianfeng Lu, and Jing-Yu Yang. In 2014 IEEE International Conference on Image Processing (ICIP), pages 2026–2030. IEEE, 2014”*

概要

本論文では、ストリートビュー画像から得られた点群を用いて 2 つのステージに分けて異なる画像間での対応付けを行なっている。最初のステージでは、出力した点群をもとに ESF-64 Description により特徴量を抽出し、Cosine similarity により特徴量ベクトルの類似度を計測している。 2 つ目のステージでは、最初のステージで得られたマッチング情報を用いて異なる画像間での点群対応づけを実施し座標変換マトリクスを出力している。対応づけには ICP を利用している。

上記 2 つのステージに分ける考え方は、本プロジェクトにおいても有効な可能性があるため、既存のアルゴリズムを一部改修することで、検証を行う。

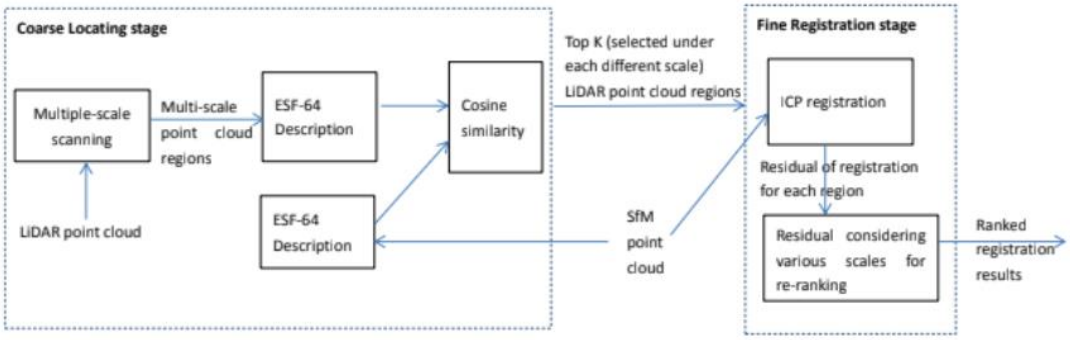


図 7-13 Street view cross-sourced point cloud matching and registration.の処理フロー図*(論文より引用)*

2. *“A systematic approach for cross-source point cloud registration by preserving macro and microstructures. Xiaoshui Huang, Jian Zhang, Lixin Fan, Qiang Wu, and Chun Yuan. IEEE Transactions on Image Processing, 26(7):3261–3276, 2017.”*

概要

本論文では、Kinect とスマートフォンのカメラなどから得られた点群を用いて 5 つのステージに分けて異なる画像間でのマッチングを行なっている。マクロ構造とミクロ構造に分けた 5 段階でのマッチングの考え方は本プロジェクトにおいても有効な可能性があるため、既存のアルゴリズムを一部改修することで、検証を行う。

① Scale normalization
各点群のスケールを正規化する。

② Macro/micro structure extraction
各点群において、マクロ構造（点群の形状や輪郭）とミクロ構造（局所的な特徴や面の構造）に分けてデータを抽出する。

③ Graph construction
②で抽出した情報をグラフ構造に置き換え、マクロ構造・ミクロ構造における特徴量やベクトルによる関係を示す。

④ Optimization
③で構築したグラフ構造をもとに各特徴点の類似度を定義する。この時、類似度が最大になるよう特徴量を最適化する。

⑤ Transformation
ICP を利用して座標変換マトリクスを出力する。

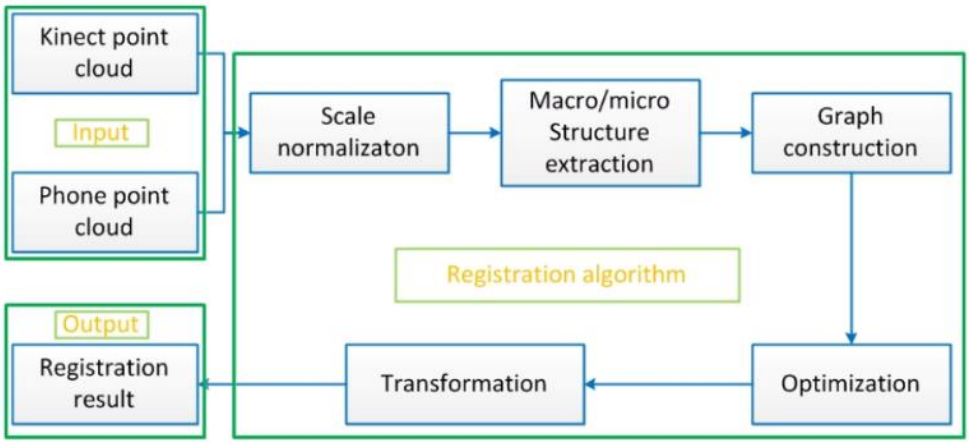


図 7-14 A systematic approach for cross-source point cloud registration by preserving macro and microstructures の処理フロー *(論文より引用)*

3. *“A coarse-to-fine algorithm for registration in 3d street-view cross-source point clouds. Xiaoshui Huang, Jian Zhang, Qiang Wu, Lixin Fan, and Chun Yuan. In 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA), pages 1–6. IEEE, 2016.”*

概要

本論文では 1 で示した「Street view cross-sourced point cloud matching and registration」と同様の考え方だが、特徴量抽出前に、Many selected regions 処理で、点群をグリッド状に分割し対応づけを実施することで高速かつ効率的に処理を実行できることを期待している。この考え方は本プロジェクトにおいても有効な可能性があるため、既存のアルゴリズムを一部改修することで、検証を行う。

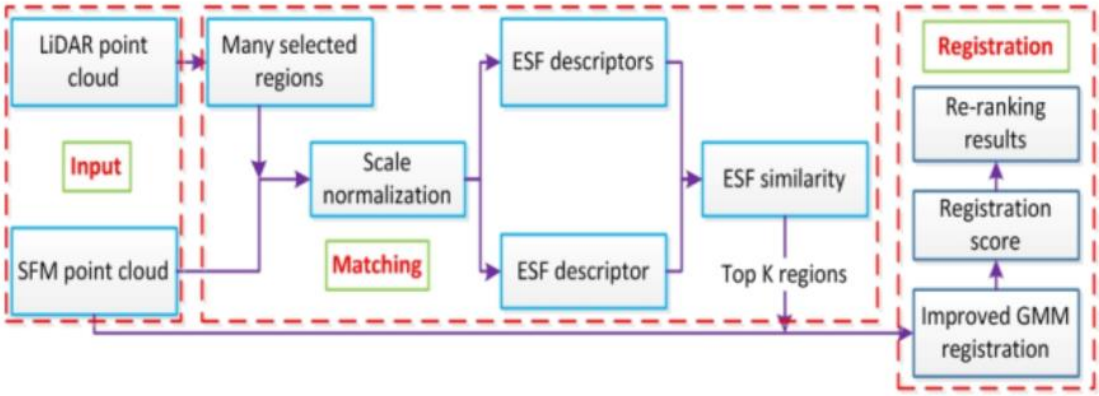


図 7-15 A coarse-to-fine algorithm for registration in 3d street-view cross-source point clouds.の処理フロー*(論文より引用)*

4. *“Fast registration for cross-source point clouds by using weak regional affinity and pixel-wise refinement. Xiaoshui Huang, Lixin Fan, Qiang Wu, Jian Zhang, and Chun Yuan. In 2019 IEEE International Conference on Multimedia and Expo (ICME), pages 1552 1557. IEEE, 2019.”*

概要

本論文では、「Weak regional affinity」及び「Pixel-wise refinement」を利用する。「Weak regional affinity」では選択された三つ組の点（トリプレット）が一致する場合のみ正しいマッチとして認識し三次元テンソルに組み込む。三次元テンソルに格納される情報は、トリプレット間の相似性に基づく。具体的には、それぞれのトリプレットについて、対応する三角形によって組み合わせられた内角の三つのコサイン値を計算することで、類似度を評価する。その後、個々の点に対して「Pixel-wise refinement」を用いて多応する Pixel 間のユーグリッド距離に基づいて類似度を評価する。

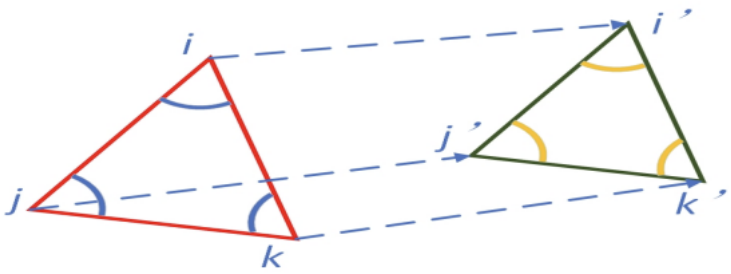


図 7-16 Weak regional affinity によるマッチングイメージ*(論文より引用)*

$$H_{ii'jj'kk'} = exp(-\|f_{ijk} - f_{i'j'k'}\|^2)$$

図 7-17 Weak regional affinity における類似度の算出式*(論文より引用)*

$$H_{ii'} = exp(-\|f_i - f_{i'}\|^2)$$

図 7-18 Pixel-wise refinement における類似度の算出式*(論文より引用)*

5. *“Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. Xiaoshui Huang, Guofeng Mei, and Jian Zhang. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11366– 11374, 2020”*

概要

本論文では、半教師あり学習を利用し高速かつ正確な対応付け処理を実現している。

他の手法では、２つの点群に対して各点ごとの対応付けを必要としているのに対し、本論文の手法では、一部の特徴的な点群について対応付けを行い、それを教師データとしてモデルを学習することで点群の座標変換マトリクスを出力する。一般的に点群の対応付けにおいては各点の対応付けに処理時間がかかるほか、さまざまな環境の変化やノイズにより対応付けの精度が低下することがある。本論文では、局所的な特徴をもとにした半教師あり学習を用いることで、対応付けの処理性能、精度が向上したことを示している。

本プロジェクトにおいても、点群 to 点群の対応付けにおいて本手法を適用することでより高速かつ高精度での対応付けを実現することが期待できる。

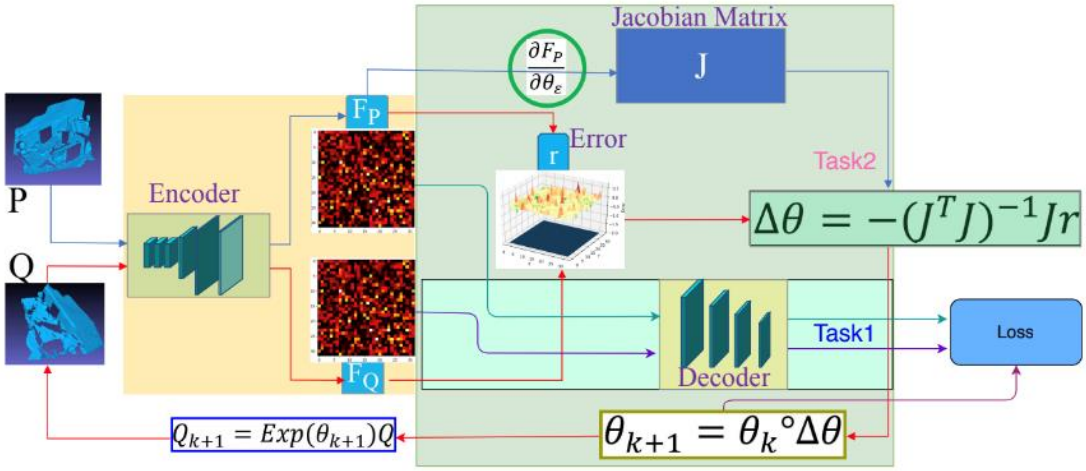


図 7-19 Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences の処理フロー*(論文より引用)*

6. *“Relative scale estimation and 3d registration of multi-modal geometry using growing least squares. Nicolas Mellado, Matteo Dellepiane, and Roberto Scopigno. IEEE transactions on visualization and computer graphics, 22(9):2160–2173, 2015.”*

概要

本論文は Growing Least Squares（GLS）法を用いて異なる解像度、スケール、ノイズレベル、データスケーリングなどを持つ複数のデバイスから取得した点群の対応付けを行っている。本プロジェクトでの課題を直接解決するものではないが、点群 to 点群対応化における精度向上に向けて GLS 法によるスケールの対応付けなどが応用できる可能性がある。

2) 【AL602】座標変換アルゴリズム

- 本アルゴリズムを利用した機能
  - 【FN508】

- アルゴリズムの詳細

PLATEAU-to-Pretia VPS で出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）へ変換する処理。本プロジェクトでは、Pretia が所有するアルゴリズムをベースに一部調整を行う。

[変換のイメージ]

点群 to 点群対応化アルゴリズムにおいて、仮に、現実世界の点群座標（x1,y1,z1）から拡大・縮小・回転・変形・移動などを計算し 3D 都市モデルの座標（x2,y2,z2）へ変換する関係を下記の 1 時式で示せたとする。

x2 = a11 * x1 + a12 * y1 + a13 * z1 + tx
y2 = a21 * x1 + a22 * y1 + a23 * z1 + ty
z2 = a31 * x1 + a32 * y1 + a33 * z1 + tz

その際の座標変換処理をマトリクスで示すと下記のようになる。
ここで、tx, ty, tz はそれぞれの軸に対する平行移動量を表す。
a11, a12, a13, a21, a22, a23, a31, a32, a33 は、それぞれの軸に対する線型変換係数を表す。
[0 0 0 1]はホモジニアス座標系を表す。

$$\begin{Bmatrix} x2 \\ y2 \\ z2 \\ 1 \end{Bmatrix} = \begin{pmatrix} a11 & a12 & a13 & tx1 \\ a21 & a22 & a23 & ty1 \\ a31 & a32 & a33 & tz1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{Bmatrix} x1 \\ y1 \\ z1 \\ 1 \end{Bmatrix}$$

7-4. データインターフェース

7-4-1. ファイル入力インターフェース

1) 【IF501】スマートフォンカメラ画像

専用のレコーダーで撮影した現地のカメラ画像データ。

- 本インターフェースを利用する機能：
  - 【FN501】
- ファイル形式
  - バイナリ形式

図 7-20 3D スキャナで撮影した画像のイメージ

2) 【IF502】CityGML 形式 3D 都市モデル

3D 都市モデルの CityGML データ。

- 本インターフェースを利用する機能：
  - 【FN503】
- ファイル形式
  - CityGML 形式

3) 【IF503】AR 表示用 3D コンテンツ

ユーザーアプリ上で AR 表示する 3D コンテンツ。

- 本インターフェースを利用した機能
  - 【FN511】
- ファイル形式
  - OBJ 形式

4) 【IF504】スマートフォンのカメラ画像

ユーザーアプリケーション上で撮影したカメラ画像。

- 本インターフェースを利用する機能
  - 【FN509】
- データ形式
  - バイナリ形式

5) 【IF505】現実世界のカメラ画像

専用のレコーダーで撮影し、Pretia サーバへ送信された現地のカメラ画像データ。

- 本インターフェースを利用する機能
  - 【FN502】
- データ形式
  - PNG 形式

6) 【IF506】3D 都市モデルの画像データ

Unity 上に取り込んだ 3D 都市モデルをもとに、Unity カメラを利用し撮影したシミュレーション画像。

- 本インターフェースを利用する機能
  - 【FN505】
- データ形式
  - PNG 形式

7) 【IF507】3D 都市モデルに最適化した補正後自己位置情報

PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で変換し取得した 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）。

- 本インターフェースを利用する機能
  - 【FN510】
- データ形式
  - バイナリ形式

8) 【IF508】スマートフォンのカメラ画像

ユーザーアプリケーション上で撮影し、Pretia サーバに送信したカメラ画像。

- 本インターフェースを利用する機能
  - 【FN507】
- データ形式
  - PNG 形式

7-4-2. ファイル出力インターフェース

1)　【IF601】画像データ

専用のレコーダーで撮影した現地のカメラ画像データ。

- 本インターフェースを利用した機能
  - 【FN501】
- データ形式
  - PNG 形式

2)　【IF602】3D 都市モデルのスナップショット画像データ

Unity シミュレーションカメラで撮影した 3D 都市モデルの画像データ。

数千枚の画像が必要なため、変換にあたっては簡易ツールを作成し、半自動で出力する。

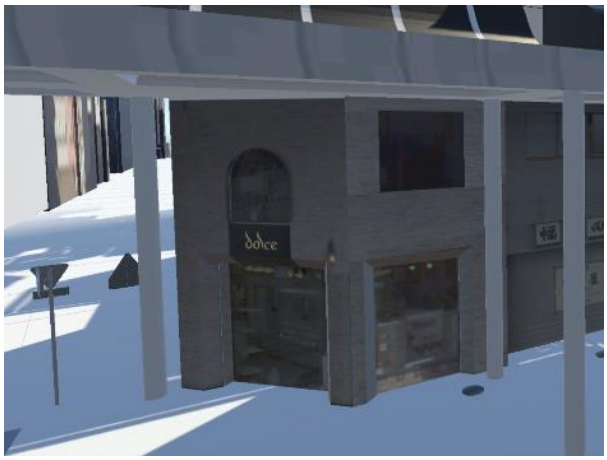


図 7-21 Unity カメラで撮影した画像のイメージ

- 本インターフェースを利用した機能
  - 【FN504】
- データ形式
  - PNG 形式

3) 【IF603】iPhone 用のインストールパッケージ

Unity 上でビルドした iPhone 用のアプリインストールパッケージ。

- 本インターフェースを利用した機能
  - 【FN512】
- データ形式
  - ipa 形式

4) 【IF604】画像データ

ユーザーアプリケーション上で撮影したカメラ画像。

- 本インターフェースを利用した機能
  - 【FN509】
- データ形式
  - PNG 形式

5) 【IF605】3D 都市モデルに最適化した補正後自己位置情報

PLATEAU-to-Pretia VPS にて出力した自己位置情報（x1, y1, z1）を点群 to 点群対応化アルゴリズムで出力したメタデータ（座標変換マトリクス）で変換し取得した 3D 都市モデルに最適化した自己位置情報（x2, y2, z2）。7 つの float 値としてユーザーアプリからの自己位置推定処理要求の返却値として出力する。

- pose data(x, y, z)：位置座標データ
- orientation(x, y, z, w)：向きや姿勢のデータ

- 本インターフェースを利用した機能
  - 【FN508】
- データ形式
  - json 形式

6) 【IF606】3D コンテンツ

Unity 上で事前に表示位置を指定した 3D コンテンツ。

- 本インターフェースを利用した機能
  - 【FN510】
- データ形式
  - OBJ 形式

7-4-3. 内部連携インターフェース

1) 【IF701】CityGML 形式 3D 都市モデル
3D 都市モデルの CityGML データ。

- 本インターフェースを利用する機能
  - 【FN504】
- ファイル形式
  - CityGML 形式

2) 【IF702】Unity プロジェクトファイル
Unity 上で開発した管理情報（yaml ファイル）とコンテンツ(OBJ, PNG ファイル)をまとめたディレクトリ一式。

- 本インターフェースを利用する機能
  - 【FN512】
- ファイル形式
  - yaml 形式, OBJ 形式, PNG 形式

3) 【IF703】現実世界から出力した点群
専用のレコーダーで撮影し、Pretia サーバへ送信された現地のカメラ画像データをもとに出力した 3D の点群データ。

- 本インターフェースを利用する機能
  - 【FN502】【FN506】【FN507】
- ファイル形式
  - json 形式

4) 【IF704】3D 都市モデルから出力した点群
Unity 上で取得した 3D 都市モデルの画像データを、SfM をベースとした VPS 用点群出力処理を用いて出力した点群データ。

- 本インターフェースを利用する機能：
  - 【FN506】
- ファイル形式

- json 形式

5) 【IF705】座標変換マトリクス

現実世界から出力した点群と 3D 都市モデルから出力した点群をもとに座標情報等を整理し双方の対応関係を導出し、点群と合わせて既存の自己位置推定アルゴリズムに処理させることのできるメタデータ（座標変換マトリクス）。

- 本インターフェースを利用する機能：
  - 【FN506】【FN508】
- ファイル形式
  - json 形式

6) 【IF706】補正前自己位置情報（x1, y1, z1）

実世界から出力した点群と、スマートフォン向け AR アプリケーションから送られたカメラ映像から出力された点群をマッチングし、自己位置情報（x1, y1, z1）として算出したデータ。

- 本インターフェースを利用する機能：
  - 【FN507】【FN508】
- ファイル形式
  - バイナリ形式

7-4-4. 外部連携インターフェース

本プロジェクトにおいて、外部とのデータ連携は実施しない。

7-5. 実証に用いたデータ

7-5-1. 活用したデータ一覧

1） 利用した 3D 都市モデル

- 年度：2020 年度
- 都市名：沼津市
- ファイル名：22203_numazu-shi_2021_citygml_4_op
- メッシュ番号：52385628, 52385618, 52385608, 52384698（インデックスマップで黄色囲いの箇所）

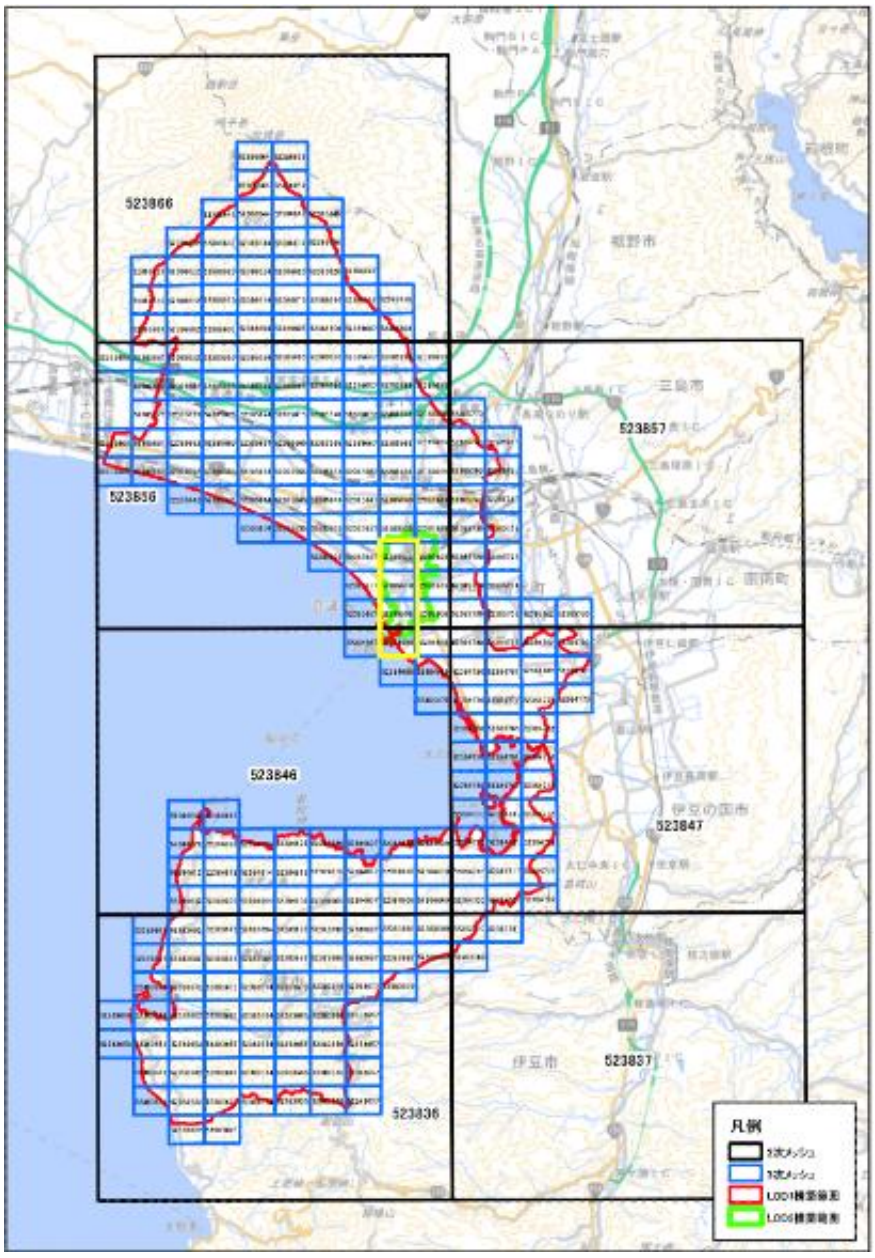


図 7-22 インデックスマップ – 全体像（沼津市）

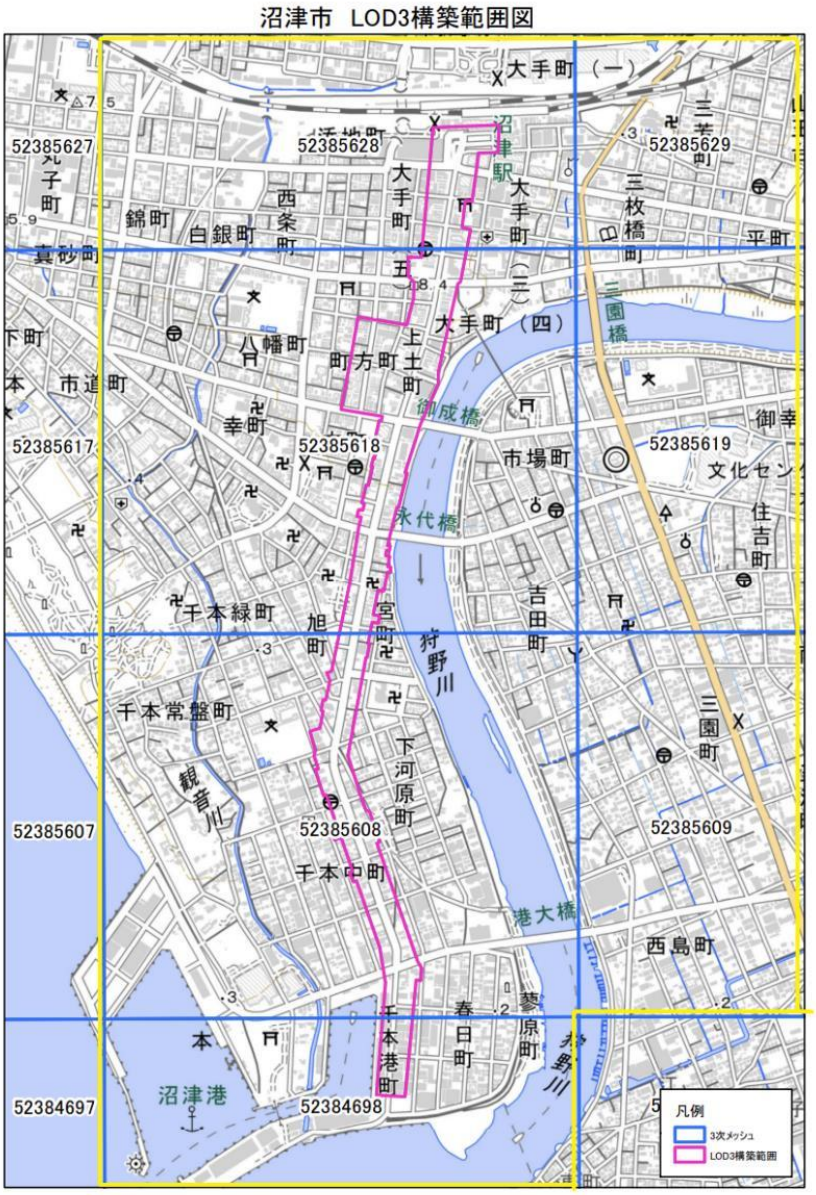


図 7-23 インデックスマップ – 詳細（沼津市）

表 7-5 利用した 3D 都市モデル

地物	地物型	属性区分	属性名	利用想定	データを利用した機能（ID）
建築物 LOD2	bldg:Building	空間属性	bldg:lod2Solid	建築物の LOD2 の立体、VPS 用マップ	FN505
	app:Appearance	空間属性	app:imageURI など	建築物の LOD2 テクスチャ、VPS 用マップ	FN505
建築物 LOD3	bldg:Building	空間属性	bldg:lod3Solid	建築物の LOD3 の立体、VPS 用マップ	FN505
	app:Appearance	空間属性	app:imageURI など	建築物の LOD3 テクスチャ、VPS 用マップ	FN505
道路 LOD3	tran:Road	空間属性	tran:lod3MultiSruface	道路の LOD3 立体、VPS 用マップ	FN505
都市設備 LOD3	frn:CityFurniture	空間属性	frn:lod3Geometry	都市設備の LOD3 立体、VPS 用マップ	FN505
	app:Appearance	空間属性	app:imageURI など	都市設備の LOD3 テクスチャ、VPS 用マップ	FN505
地形 LOD2	dem:TINRelief	空間属性	dem:tin	地形の LOD2 立体、VPS 用マップ	FN505

2) 利用したその他のデータ

表 7-6 利用したその他データ（一覧）

ID	エリア（都市）	活用データ	内容	データ形式	更新情報	出所	データを利用した機能（ID）
DT501	-	スマートフォン GPS	スマートフォンの GPS で取得した現在地座標	地理座標の数値	-	端末	FN507
DT502	-	スマートフォン IMU	スマートフォンのセンサで取得した加速度と角速度	数値	-	端末	FN507

7-5-2. 生成・変換したデータ

表 7-7 生成・変換したデータ

ID	システムに入力するデータ（データ形式）	用途	処理内容	データ処理ソフトウェア	活用データ（データ形式）	データを利用した機能（ID）
DT601	3D 都市モデル（CityGML 形式）	点群出力アルゴリズム及び Unity での開発で利用する	● 3D 都市モデル（CityGML）から、PLATEAU SDK for Unity で Unity 上へ取り込む	PLATEAU SDK for Unity	3D 都市モデル（PNG 形式）	FN505
DT602	3D 都市モデル（PNG 形式）	点群 to 点群対応化アルゴリズム及び自己位置推定処理で利用する	● 3D 都市モデルからの点群出力アルゴリズムで点群データへ変換する	3D 都市モデルからの点群出力アルゴリズム	3D 都市モデルから出力した点群(json 形式)	FN505
DT603	現実世界のカメラ画像（PNG 形式）	点群 to 点群対応化アルゴリズム及び自己位置推定処理で利用する	● 現実世界からの点群出力アルゴリズムで点群データへ変換する	現実世界からの点群出力アルゴリズム	現実世界から出力した点群(json 形式)	FN502
DT604	3D 都市モデルから出力した点群	座標変換アルゴリズムで現実空間の座標情報を 3D ファイルの座標情報に変換する	● 点群 to 点群対応化アルゴリズムで座標変換マトリクスへ変換する。	点群 to 点群対応化アルゴリズム	座標変換マトリクス(json 形式)	FN506
DT605	現実世界から出力した点群					
DT606	現実世界から出力した点群(json 形式)	座標変換アルゴリズムのインプット情報として利用する	● PLATEAU-to-Pretia VPS（自己位置推定アルゴリズムで補正前自己位置情報へ	PLATEAU-to-Pretia VPS（自己位置推定アルゴリズ	補正前自己位置情報[pose data (x, y, z) and orientation (x, y, z, w) 7 float]	FN507
DT607	ユーザーアプリケーションを利用して撮					

	影した画像		変換する	ム）		
DT608	補正前自己位置情報	ユーザーアプリにおいてコンテンツ表示する際の自己位置情報として利用する	● 座標変換処理で補正後自己位置情報へ変換する	座標変換アルゴリズム	補正後自己位置情報[pose data (x, y, z) and orientation (x, y, z, w) 7 float]	FN508
DT609	座標変換マトリクス(json 形式)					

7-6. ユーザーインターフェース

7-6-1. 画面一覧

1） ユーザーアプリ（スマートフォン）での表示画面

表 7-8 PC 画面一覧

ID	連携（ID）	画面名	画面説明	画面を表示した機能（ID）
SC501	SC502	メイン画面	● 利用するエリアを指定しスキャン画面へ繊維する。	FN507, FN508
SC502	SC501	3D スキャン及び AR コンテンツ表示画面	● 現実世界の画像をスキャンし Pretia for PLATEAU サーバへ画像データを送信する。 ● Pretia for PLATEAU サーバから取得した自己位置情報をもとに AR コンテンツを表示する。	FN507, FN508

7-6-2. 画面遷移図

1) スマートフォン用画面

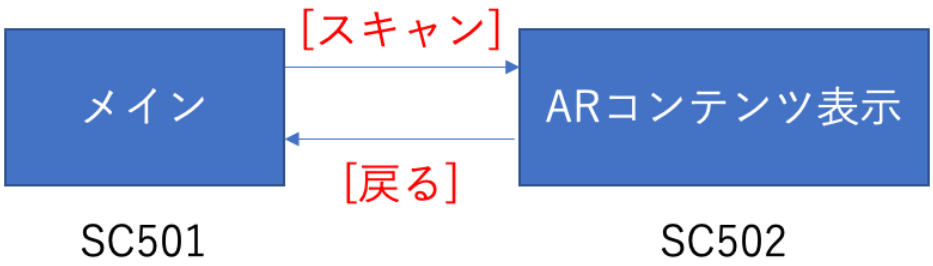


図 7-24 スマートフォン用画面遷移図

7-6-3. 各画面仕様詳細

1) スマートフォン用画面

1. 【SC501】メイン画面

- 画面の目的・概要
  - 利用するエリアを指定しスキャン画面へ遷移する。

- 画面イメージ

図 7-25 メイン画面のイメージ

2. 【SC502】3D スキャン及び AR コンテンツ表示画面

- 画面の目的・概要
  - 現実世界の画像をスキャンし Pretia for PLATEAU サーバへ画像データを送信する。
  - Pretia for PLATEAU サーバから取得した自己位置情報をもとに AR コンテンツを表示する。
- 画面イメージ

図 7-26 3D スキャン及び AR コンテンツ表示画面のイメージ

7-7. 実証システムの利用手順

7-7-1. 実証システムの利用フロー

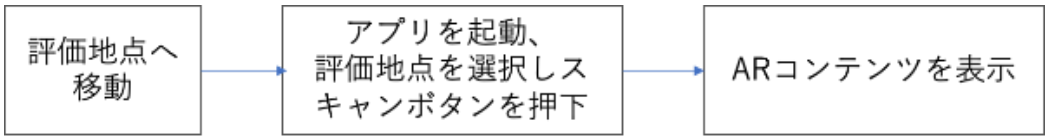


図 7-27 システムの利用フロー

- 利用フローは以下のとおり
  1. 対象とする 10 か所の評価地点へ移動
  2. アプリケーションの起動
  3. アプリケーションの機能によりカメラを起動し周囲をスキャンし AR コンテンツ（建物の 3D 都市モデル）を表示
  4. 現実の建物との位置のずれを計測し精度を評価する。

7-7-2. 各画面操作方法

1） 評価地点へ移動

- 本システムは評価地点の画像情報をもとに自己位置を推定し AR コンテンツを表示する。そのため、利用者はシステム利用時に評価ポイントとして設定された場所に移動する必要がある。

図 7-28 評価地点でのアプリ利用イメージ

2) アプリの起動

- ユーザーアプリは以下の手順で利用する。
  - アプリを起動する。
  - 評価地点を選択する。
  - 「スキャン開始」ボタンを押下する。

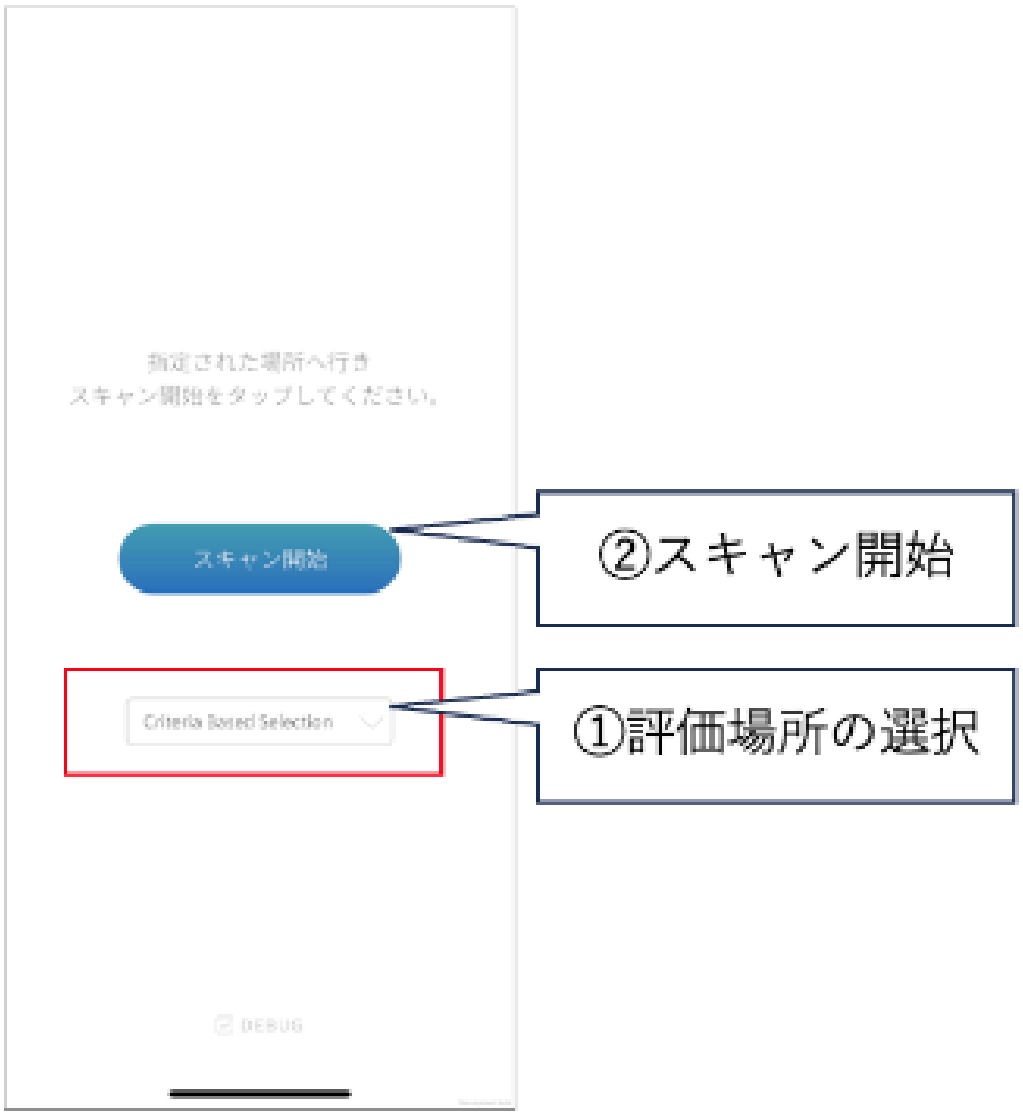


図 7-29 ユーザーアプリメイン画面の利用イメージ

3) AR コンテンツの表示

- 「スキャン開始」ボタン押下後、スマートフォンカメラで周囲を見渡し自己位置推定処理の完了を待つ。完了後、上部に表示のステータスが[Found]へ遷移し AR コンテンツが同画面上に表示される。

図 7-30 スキャンから AR コンテンツ表示までのイメージ

8. スマートフォン向けアプリケーション：実証技術の検証

8-1. アルゴリズムの検証

8-1-1. 検証目的

- 3D 都市モデルからの点群出力及び点群 to 点群対応化の処理精度及び処理性能の評価によって、本アルゴリズムの優位性を検証する。

8-1-2. KPI

表 8-1 KPI 一覧

No.	評価指標・KPI	目標値	目標値の設定理由	検証方法サマリー
1	3D 都市モデルからの点群出力可否	可	● 昨年度までの取組において、3D 都市モデルから点群が出力できないという事象があったため	● 点群出力結果の目視確認
2	対応化の変換エラー値	30cm 以下	● 3D 都市モデルの VPS 活用において 3D 都市モデルから出力した点群と現実世界から出力した点群の対応化精度をどれだけ高められるかが最も重要な指標となる。	● 点群 to 点群対応化の変換エラー値の算出
3	対応化処理の所要時間の測定可否	測定できる	● 実用化に向けて点群 to 点群がリアルタイムに処理できることが重要となる。本プロジェクトスコープでは性能値は KPI 対象外だが、将来の KPI 設定に向けて現時点での処理性能は把握することとする。	● 点群 to 点群対応化処理性能をプログラム上で検出

8-1-3. 検証方法と検証シナリオ

- 検証方法

1) 今回の実証実験における対象エリアである「沼津駅～沼津港」のデータに対して、スマートフォン向け VPS 技術の確立に必要な VPS マップの作成が 3D 都市モデルを活用して実現できるのかを確認する。

2) 3D 都市モデルを活用して作成した VPS マップと現実世界の点群の対応化を行うことで自己位置推定を実現するために、点群対応化機能の位置精度を既存のデータセットと比較することで、アルゴリズム自体の優位性を確認する。

3) 上記の対応化機能のユーザー検証に向けて処理時間自体を計測可能か確認する。

- 検証シナリオ

1) 3D 都市モデルからの点群出力可否
   - 3D 都市モデル(LOD3)を Unity 上の開発環境に取り込み、Unity の物理カメラ機能をもとに出力した画像を【FN501】3D 都市モデルからの点群出力処理で点群データへ変換できることを確認する。結果は出力した点群データを 3D 都市モデルと目視で比較しておよその形状が認識できることを確認する。

2) 対応化の変換エラー値計測
   - 点群 to 点群対応化アルゴリズムでは、3D 都市モデルから出力した点群と現実世界から出力した点群の 2 つの異なるソースデータをもとにした点群の対応化を行う。

※評価対象のデータセットについて
   - 本アルゴリズムの精度を評価するにあたり、2 つの異なるソースデータ間における対応化すべき点の正しい組み合わせをあらかじめセットする必要がある。
   - 一方でこれらを対象となる 3D 都市モデル及び現実世界から出力した点群に対して手動で実施することは非常に困難であるため、今回の実証実験では、公開されているオープンデータ（3D Cross-source point cloud registration (3DCSR) benchmark ）を活用することで、アルゴリズムの精度を評価することとした。

3) 対応化処理の所要時間が測定可否
   - 点群 to 点群対応化アルゴリズムにおいて、全体の処理時間をログとして出力する機能を実装することで、対応化処理の所要時間を取得した。本プロジェクトスコープでは性能値は KPI 対象外だが、将来の KPI 設定に向けて現時点での処理性能は把握することを目的としてデータを取得する。

8-1-4. 検証結果

- 検証結果まとめ
  - 3D 都市モデル（LOD3）から点群を出力し誤差 30cm 以下で現実世界からの点群と 3D 都市モデルからの点群を処理時間測定可能な状態で対応化することに成功した。
  - 点群出力においては、3D 都市モデルのテクスチャの品質を向上することでより適切な点群出力が可能であることが示された。

表 8-2 検証結果サマリー

黄セル：KPI 達成	青セル：KPI 未達

No.	評価指標・KPI	目標値	結果		示唆
			機械学習による対応化	データ処理による対応化	
1	3D 都市モデルからの点群出力可否	可	可		● 沼津エリアの LOD3 データをもとに点群が出力できることを確認した ● 一方、同じ LOD3 であっても横浜エリアのように樹木や建物などのテクスチャに特徴がない部分が多いエリアでは適切な点群出力ができなかった。このことから、本技術の適用には 3D 都市モデル上のテクスチャの線や色の変化を明確にする必要があると考えられる
2	対応化の変換エラー値	30cm 以下	1.89cm	1.00cm	● 機械学習のデータとして対応する点群データだけでなく、座標変換マトリクスを読み込むことで、各点の平行移動に対する平均的な変換エラーは 1.89cm と目標を大きく超える成果を上げることができた ● 一方、3D 点群においては、点群全体の回転方向に対する誤差を考慮する必要性がある。評価の結果、機械学習によるアプローチでは平行移動に対する変換エラーが小さくても、回転方向に対する誤差は 30° 以上と、大きくエラーが発生していることが判明した。機械学習においてサンプルとして利用

					する点の数などのパラメータを操作したが結果に大きな違いは現れなかった。データ処理によるアプローチでは、回転方向に対する誤差も 5° 未満と比較的高精度での対応化に成功した
3	対応化処理の所要時間の測定可否	可	可	可	● 対応化処理の所要時間は平均 27 ミリ秒であった ● 自己位置推定全体の処理時間は 10 秒程度を想定しているため、現時点では対応化処理の所要時間が本技術の大きな課題となることはないと考えられる

8-2. システムの検証

8-2-1. 検証目的

- 自己位置推定の処理結果をもとに AR アプリケーションで表示した AR コンテンツの表示位置と現実世界で期待する位置との誤差及び自己位置推定に必要とする処理時間を評価することで、本システムの技術的優位性を評価する。

8-2-2. KPI

表 8-3 KPI 一覧

No.	評価指標・KPI	目標値	目標値の設定理由	検証方法サマリー
1	自己位置推定の精度	1m 未満	● 下記 2 つの観点からユーザーが AR コンテンツを利用する際の許容可能な誤差として 1m を設定 ➢ 今回の実証実験のベースシステムとして利用する VPS システムの位置精度誤差は自己位置推定に適した環境において数 cm 程度である一方、適さない環境下では数 m 程度の誤差が発生するケースが存在 ➢ 今回の実証実験では 2 つの異なる環境（現実世界と 3D モデル）の画像から出力した点群データをもとに自己位置推定を行う難易度の高い取組であり、昨年までの実証においても自己位置推定処理を実施できない場所も多く存在	● エンドユーザー向けの AR ナビゲーションを想定した簡易アプリケーションを開発 ● 3D 都市モデルのうち LOD3 データ(テクスチャあり)を持つ建築物を AR 表示し、実際の建物とのずれを測定

2	自己位置推定の処理性能	10 秒以内	● 今回の実証実験のベースシステムとして利用する VPS システムの処理性能相当として自己位置推定の処理性能に対する KPI は 10 秒以内と設定	● 開発するエンドユーザー向けの AR アプリケーションで自己位置推定に要した処理時間を取得できる機能を実装し確認

8-2-3. 検証方法と検証シナリオ

- 検証方法
  - 静岡県沼津市の沼津駅前から沼津港までの約 2km のルートを対象とし、LOD3 の 3D 都市モデルが整備されたエリアのうち 10 か所を選定
  - スマートフォン向けの AR アプリケーション上で 3D 都市モデルを用いて正しく自己位置推定が行えるか、表示される AR コンテンツの位置精度を評価する。

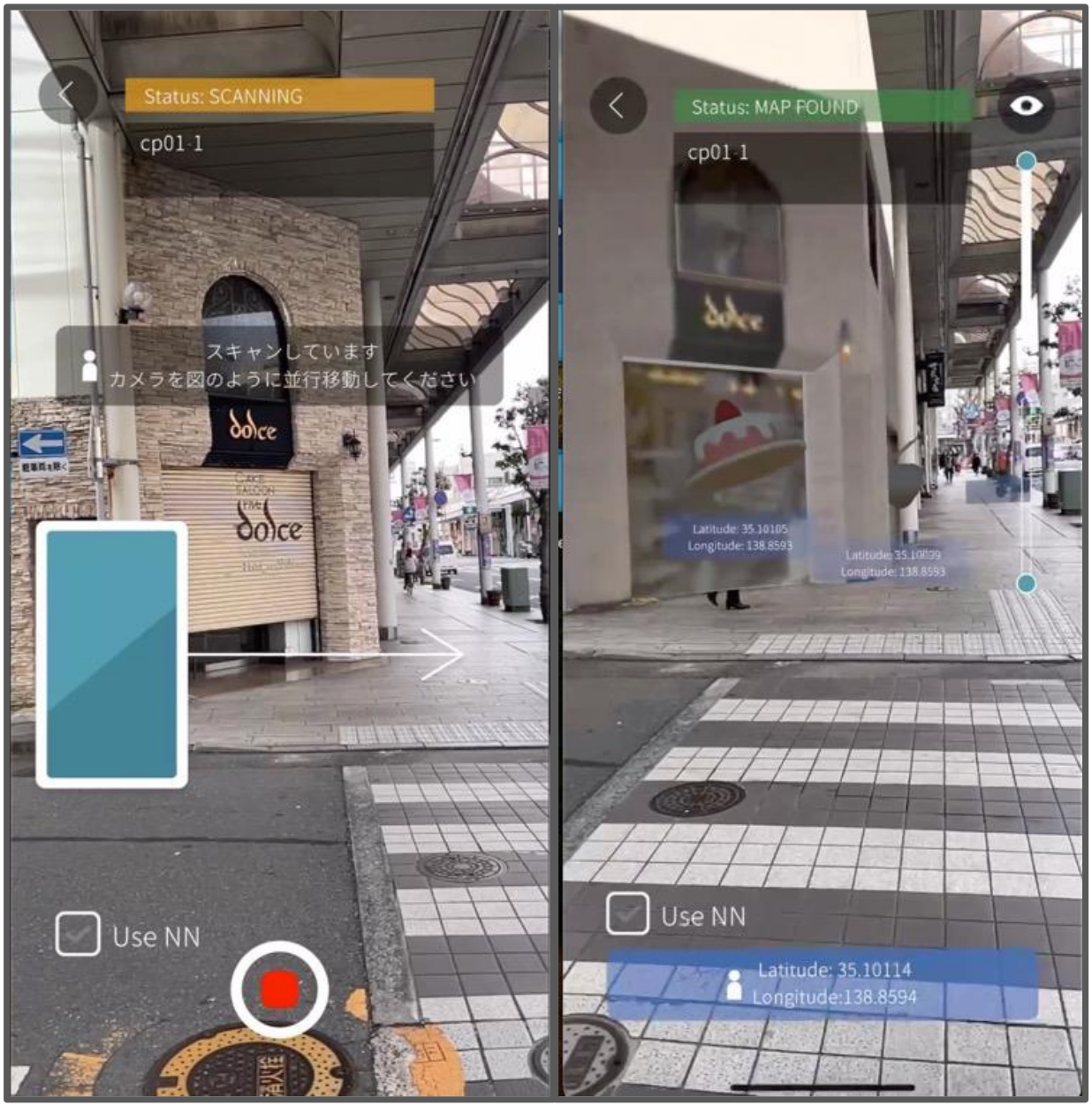


図 8-1 AR コンテンツの表示イメージ（左：投影前、右：投影後）

- 検証シナリオ

1） 自己位置推定の精度

- 自己位置推定の精度の検証においては、現実世界の指定した位置から AR コンテンツまでの距離を計測する必要がある。そのため、AR コンテンツには 3D 都市モデルの建築物モデルを採用し、現実世界の同一の建築物とのずれを定点的に計測することで検証する。
- 評価のため、AR コンテンツの透過度を柔軟に操作できる機能を実装し、スマートフォンの画面から 3D 都市モデルの建築物モデルと現実世界の同一の建築物とを比較し検証する。

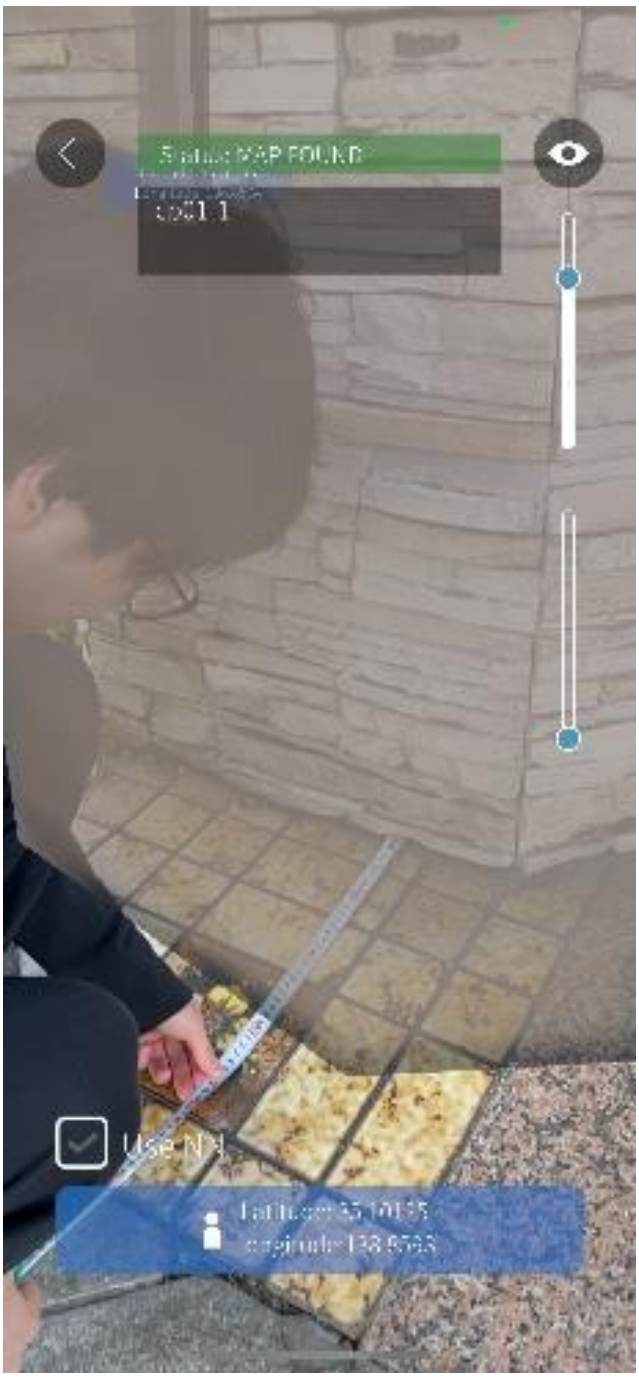


図 8-2 精度の評価イメージ

- また、AR コンテンツの表示に際しては、三次元空間内での平行移動のみではなく、回転方向にもずれが発生する場合がある。そのため、選定した各エリアにおいて定点的な 2 か所の位置誤差を算出しその平均値をそのエリアにおける位置誤差とする。

2) 自己位置推定の処理性能

- 自己位置推定の処理性能の検証においては、アプリケーション内で処理ステータスの遷移時間を測定し評価する。
- 処理ステータスは下記 4 段階で整理される。本評価では、これら 4 つのステータスの処理にかかった合計時間で評価を行う。

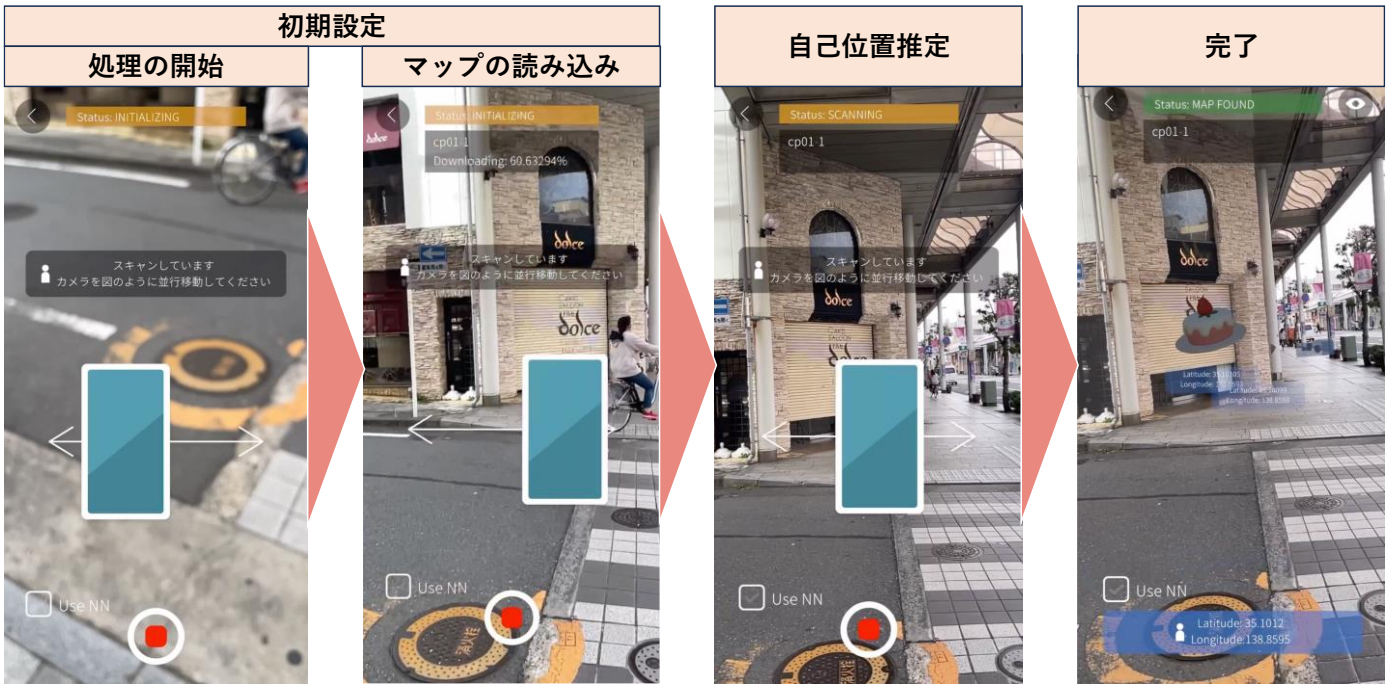


図 8-3 AR アプリケーションにおける自己位置推定処理の流れ

- 検証エリア

静岡県沼津市の沼津駅前から沼津港までの約 2km のルートを対象とし、LOD3（テクスチャあり）の建築物モデルが整備されたエリア 10 か所で評価を実施する。

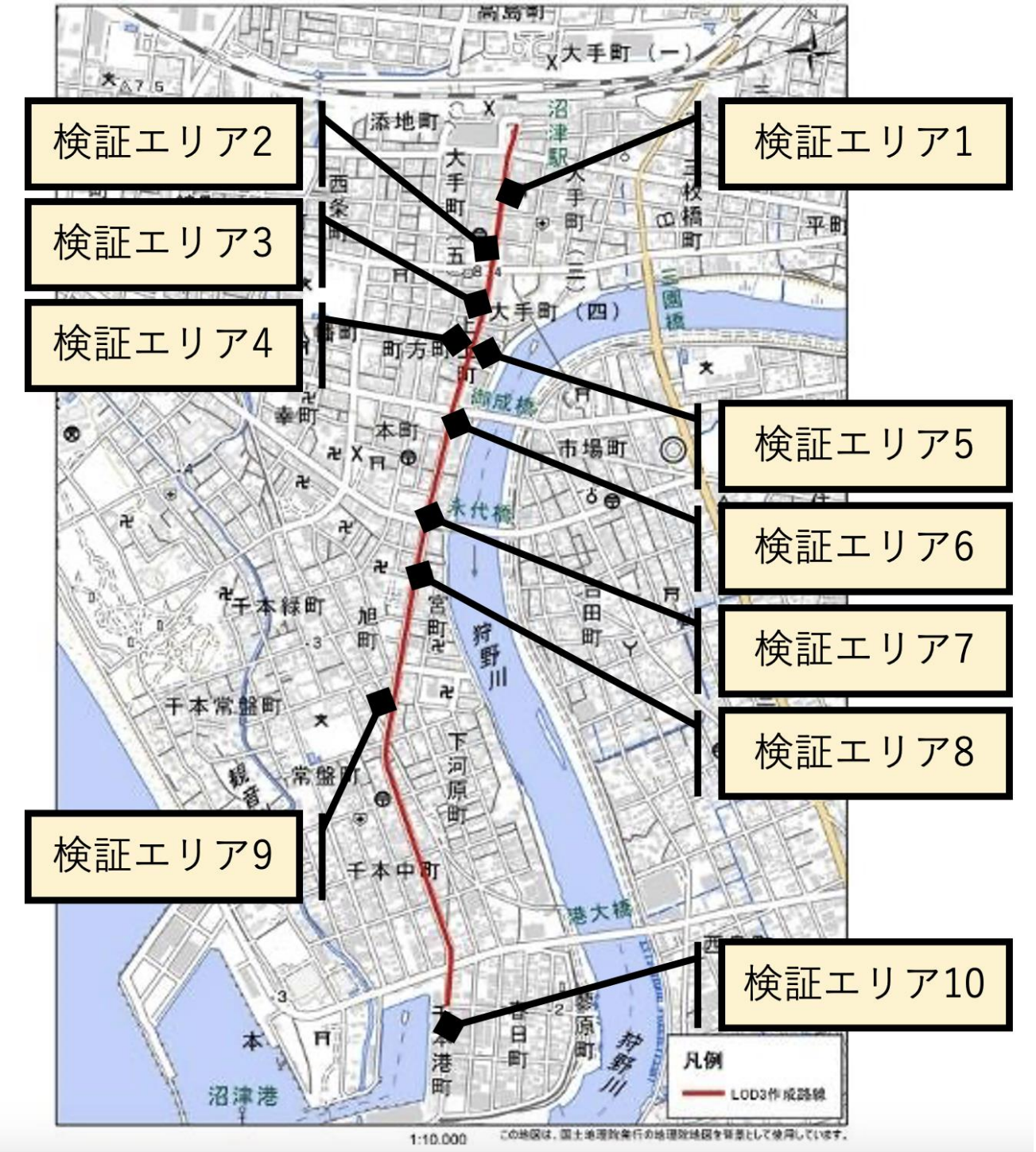


図 8-4 検証エリア

評価を実施する 10 か所の検証エリア選定においては、画像をもとに位置情報を VPS の特性から得意とする特徴的な外壁を持つ建物の周辺や、逆に苦手とする、特徴の少ない建物、一定の規則的な模様の外壁を持つ建物の周辺、周辺環境の変化が激しいポイントなどを基準に選定した。

- 検証エリア 1 ：ケーキショップ Dolce 周辺
  - 周辺イメージ

図 8-5 検証エリア 1 のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
    - 場所によっては VPS の苦手な大きなガラス窓に覆われた面あり
  - 緯度経度
    - 35.10106084865549, 138.8592531759575

- 検証エリア 2：くわはら矯正歯科医院周辺
  - 周辺イメージ

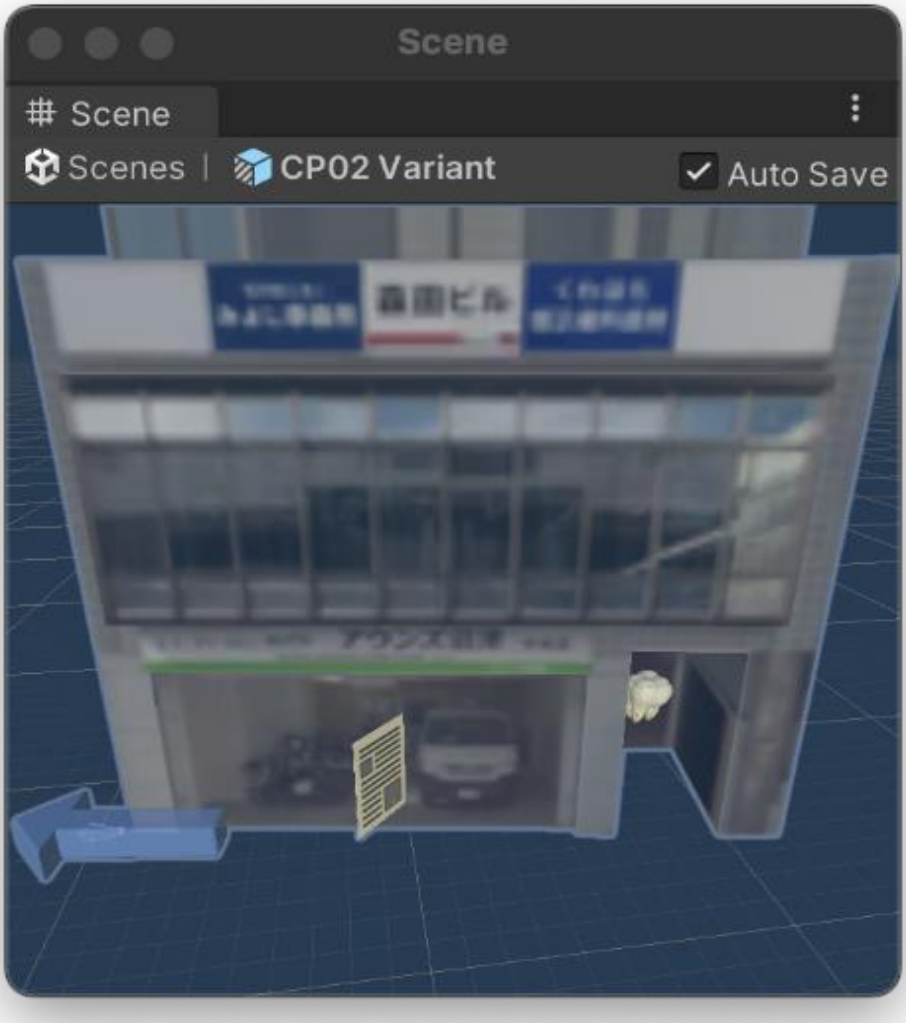


図 8-6 検証エリア 2 のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
  - 緯度経度
    - 35.09945735906196, 138.85880609201678

- 検証エリア３：魚ぶん周辺
  - 周辺イメージ

図 8-7 検証エリア３のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
    - 店舗前の看板など、小さな変化が頻繁に発生する
  - 緯度経度
    - 35.09841355738668, 138.85852557758844

- 検証エリア 4 ：デザートレストラン Grandma 上土本店周辺
  - 周辺イメージ

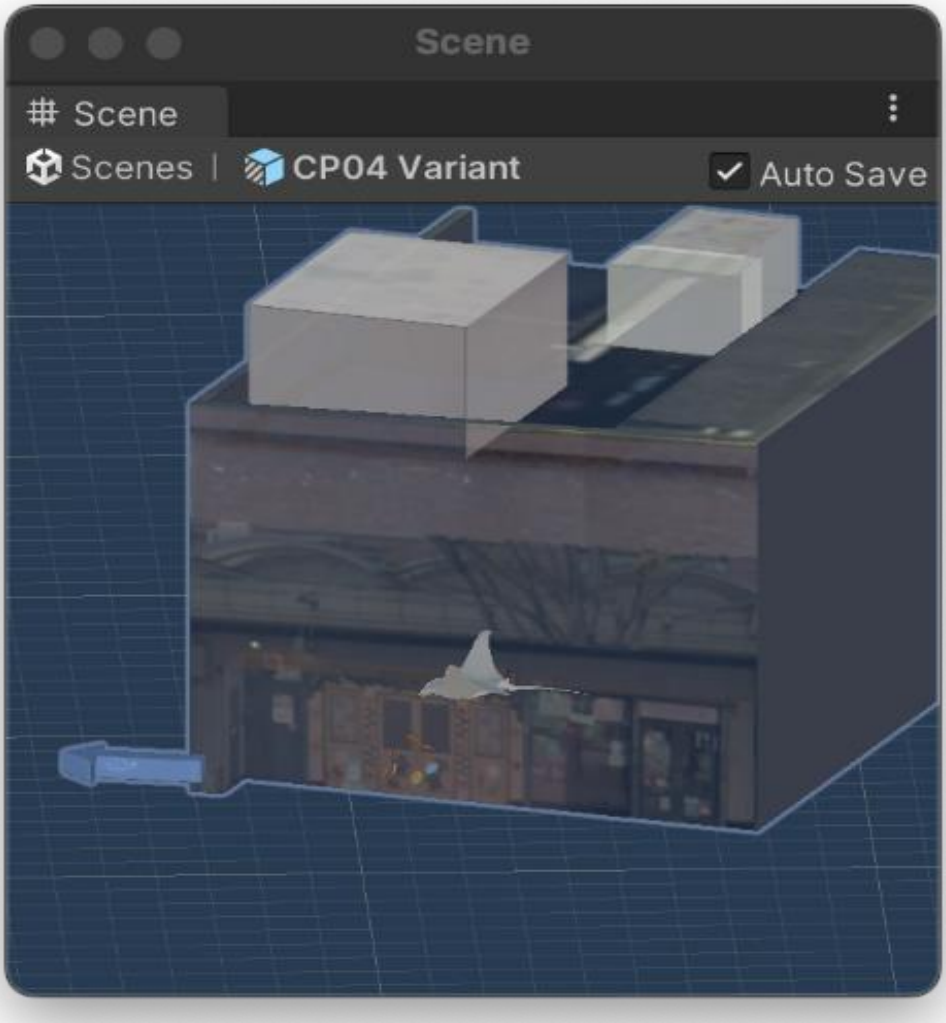


図 8-8 検証エリア 4 のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
  - 緯度経度
    - 35.09756870601267, 138.85818714463406

- 検証エリア 5 ：美容室　GUK 周辺
  - 周辺イメージ

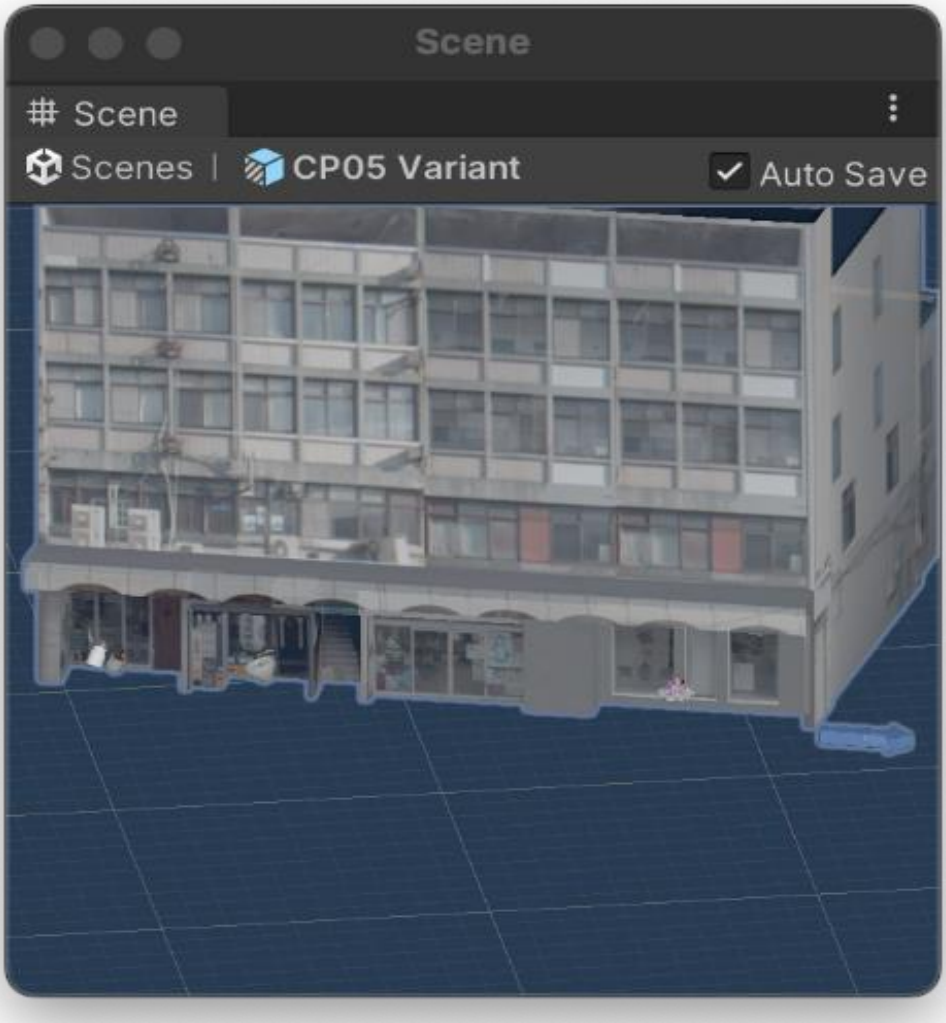


図 8-9 検証エリア 5 のコンテンツ表示イメージ

  - 選定観点
    - 白基調の特徴の少ない壁面が続くやや難易度の高いエリアとして選定
  - 緯度経度
    - 35.09760822219463, 138.85840564254767

- 検証エリア 6 ：サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺
  - 周辺イメージ

図 8-10 検証エリア 6 のコンテンツ表示イメージ

  - 選定観点
    - 外壁を長方形のシンプルなパターンで覆われており、特徴が少なく難易度の高いエリアとして選定
  - 緯度経度
    - 35.09623801128001, 138.85787273065492

- 検証エリア 7 ：（有）一樹商事周辺
  - 周辺イメージ

図 8-11 検証エリア 7 のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
    - 窓にも文字が記載されており、より難易度が低いことを想定し選定
  - 緯度経度
    - 35.09399012141429, 138.85722888886397

- 検証エリア８：不動院周辺
  - 周辺イメージ

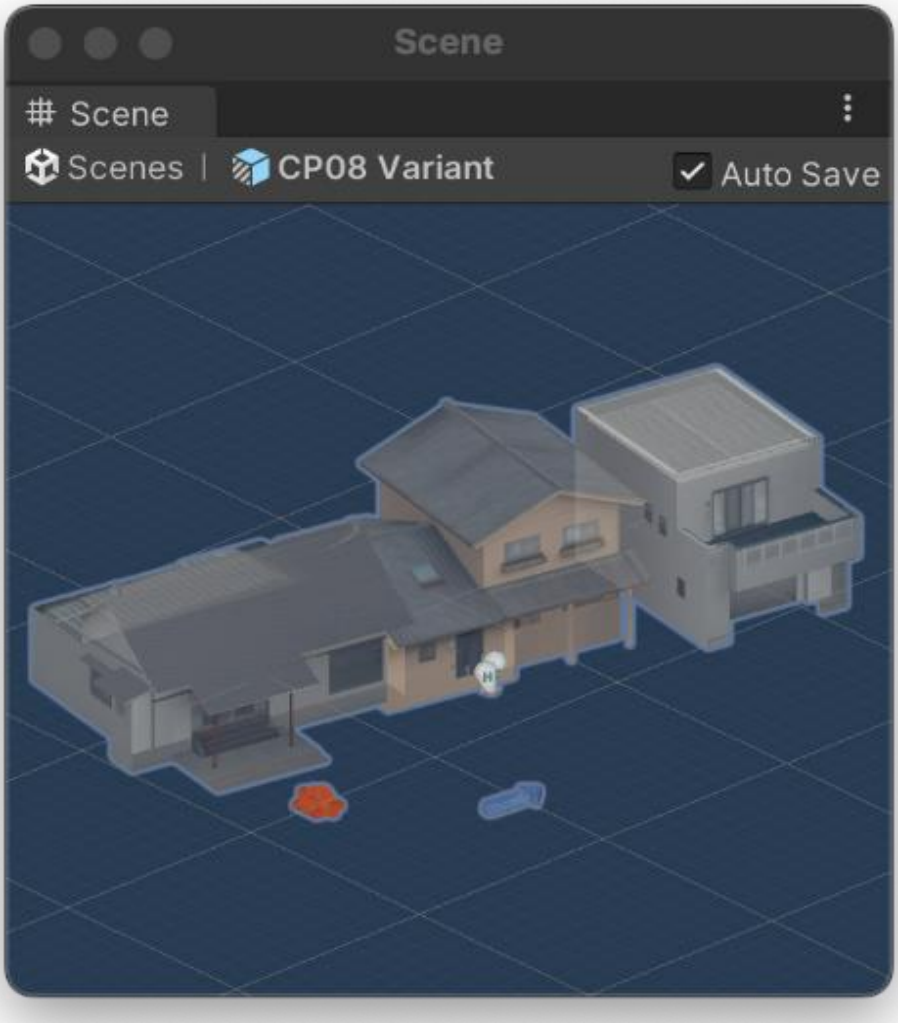


図 8-12 検証エリア８のコンテンツ表示イメージ

  - 選定観点
    - 道路から建物まで若干距離があり、特徴を掴みにくくやや難易度の高いエリアとして選定
  - 緯度経度
    - 35.0924285, 138.856854

- 検証エリア 9 ：ファミリーマート 沼津下河原町店周辺
  - 周辺イメージ

図 8-13 検証エリア 9 のコンテンツ表示イメージ

  - 選定観点
    - ✧ 広い駐車場に囲まれており道路から建物までの距離がある、かつ周辺の建物と距離もあるため、特徴が掴みづらく難易度が高いエリアとして選定
    - ✧ 駐車場内の車の停車状況が頻繁に変化するため、難易度が高いエリアとして選定
  - 緯度経度
    - ✧ 35.0900625, 138.8562446

- 検証エリア 10：（有）するが水産周辺
  - 周辺イメージ

図 8-14 検証エリア１０のコンテンツ表示イメージ

  - 選定観点
    - 外壁や看板など特徴の多い建物が多く、自己位置推定しやすいエリアとして選定
  - 緯度経度
    - 35.083128756830035, 138.85768457330707

8-2-4. 検証結果

1. 自己位置推定の精度

1)　検証結果まとめ

評価の結果、機械学習によるアプローチ、データ処理によるアプローチともに、現場のデータでは精度測定不能となる 10m 以上の誤差が発生することがわかった。アルゴリズムの評価で利用したテストデータはノイズも少なく整形されたデータであったのに対し、3D 都市モデルと現実世界からカメラで取得した点群では、アングルやデータ範囲に大きな差があったためと考えられる。実際に、表 8-4 に示すとおり、データ処理によるアプローチに一部手動のデータ補正を加えることで精度が大きく向上することが確認できた。

表 8-4 検証結果一覧（一部手動補正を含むデータ処理によるアプローチ）

NO.	検証エリア	AR コンテンツの表示誤差（cm） ※5cm 単位で計測	示唆
1	ケーキショップ Dolce 周辺	60	● 期待どおり、高精度で自己位置推定できた ● 難易度が高いと予測していたガラスの多い面も、壁面のエアコン室外機等をポイントに自己位置推定できた ● 店舗のシャッター開閉に関わらず自己位置推定できた
2	くわはら矯正歯科医院周辺	20	● 期待どおり、高精度で自己位置推定できた ● シャッターの開閉に関わらず自己位置推定できた ● 車庫内の車の有無に関わらず自己位置推定できた
3	魚ぶん周辺	20	● 期待どおり、高精度で自己位置推定できた ● 看板の位置のズレなどの影響もなく自己位置推定できた ● 店舗の営業有無（看板や暖簾の有無）に関係なく自己位置推定できた
4	デザートレストラン Grandma 上土本店周辺	35	● 期待どおり、高精度で自己位置推定できた ● 扉の開閉に関わらず自己位置推定できた
5	美容室　GUK 周辺	30	● 壁面に大きな特徴はなかったものの、看板や窓枠などをポイントに比較的高精度に自己位置推定できた
6	サンフロント静岡新聞ＳＢＳ静岡放送東部総	160	● 当初の想定どおり、外壁を長方形のシンプルなパターンで覆われているため、高精度

	局ビル周辺		での自己位置推定には至らなかった
7	（有）一樹商事周辺	40	● 期待どおり、高精度で自己位置推定できた ● 窓内の文字等を特徴点として捉えることを期待していたが、エアコンの室外機や窓枠などを優先的に特徴として得ることで自己位置推定を実現していた
8	不動院周辺	40	● 道路から建物まで若干距離があり、特徴を掴みにくくやや難易度の高いと想定していたが、建物の梁などを中心に特徴点を抽出し高精度で自己位置推定できた
9	ファミリーマート 沼津下河原町店周辺	-	● 自己位置推定処理は成功したものの、測定不能な精度誤差が発生した ● 当初の想定どおりではあるものの、広い駐車場に囲まれており道路から建物までの距離がある、かつ周辺の建物と距離もあるため、共通の特徴が掴めず適切に自己位置推定できなかったと考える
10	（有）するが水産周辺	70	● 人通りも多く環境変化が大きかったため、自己位置推定は期待よりは低い精度となったが、目標値はクリアできた

● 手動補正を含むデータ処理によるアプローチ

前述のとおり、機械学習によるアプローチ、データ処理によるアプローチともに、現場のデータではノイズなどの影響から精度誤差測定不能となる 10m 以上の非常に大きな誤差が発生することがわかった。

図 8-15 機械学習によるアプローチでの精度イメージ

よって、現場でのシステムの評価ではデータ処理によるアプローチに対して、一部手動でのデータ補正を行う方式を採用し、将来的な技術の実現性を評価することとした。以下に手動によるデータ補正を含むデータ処理による対応化アルゴリズムの概要を記載する。本アルゴリズムでは、点群 to 点群対応化 Step1 で、手動により 4 点、対応化すべき点を指定する。

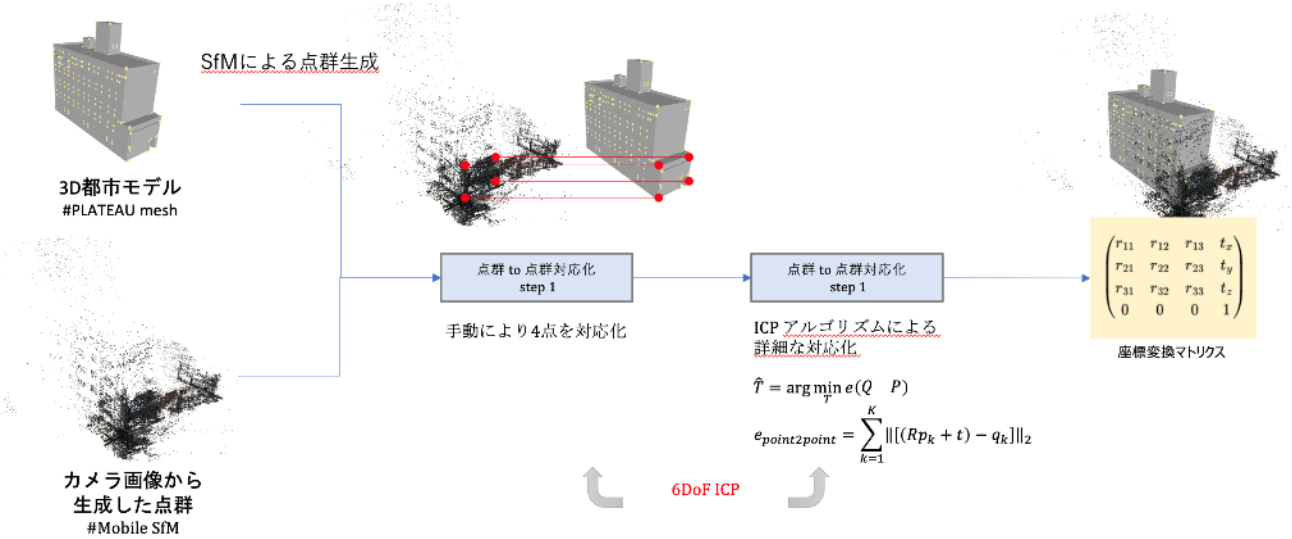


図 8-16 手動でのデータ補正を伴う点群 to 点群対応化の処理イメージ

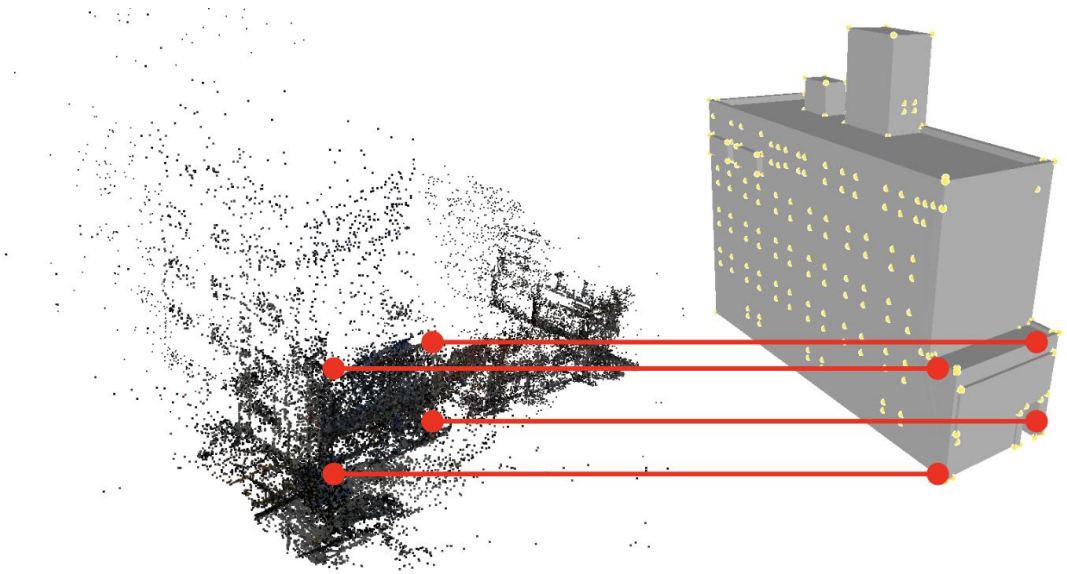


図 8-17 手動での対応化すべき 4 点の指定イメージ

- 現場のデータノイズについて
  ① 現実世界から取得した点群は、カメラから遠い場所になればなるほど、点群の密度が極端に低くなり、データ品質が低下する。
  ② 現実世界から取得した点群は、ユーザーが現地でカメラを向けた範囲のみ取得できるため、建物の高層階や背面の情報がノイズとなる。
  ③ 3D 都市モデルは建物ごとにデータ管理するため、周辺の建物をどの範囲まで対応化対象とするか判断が難しい。

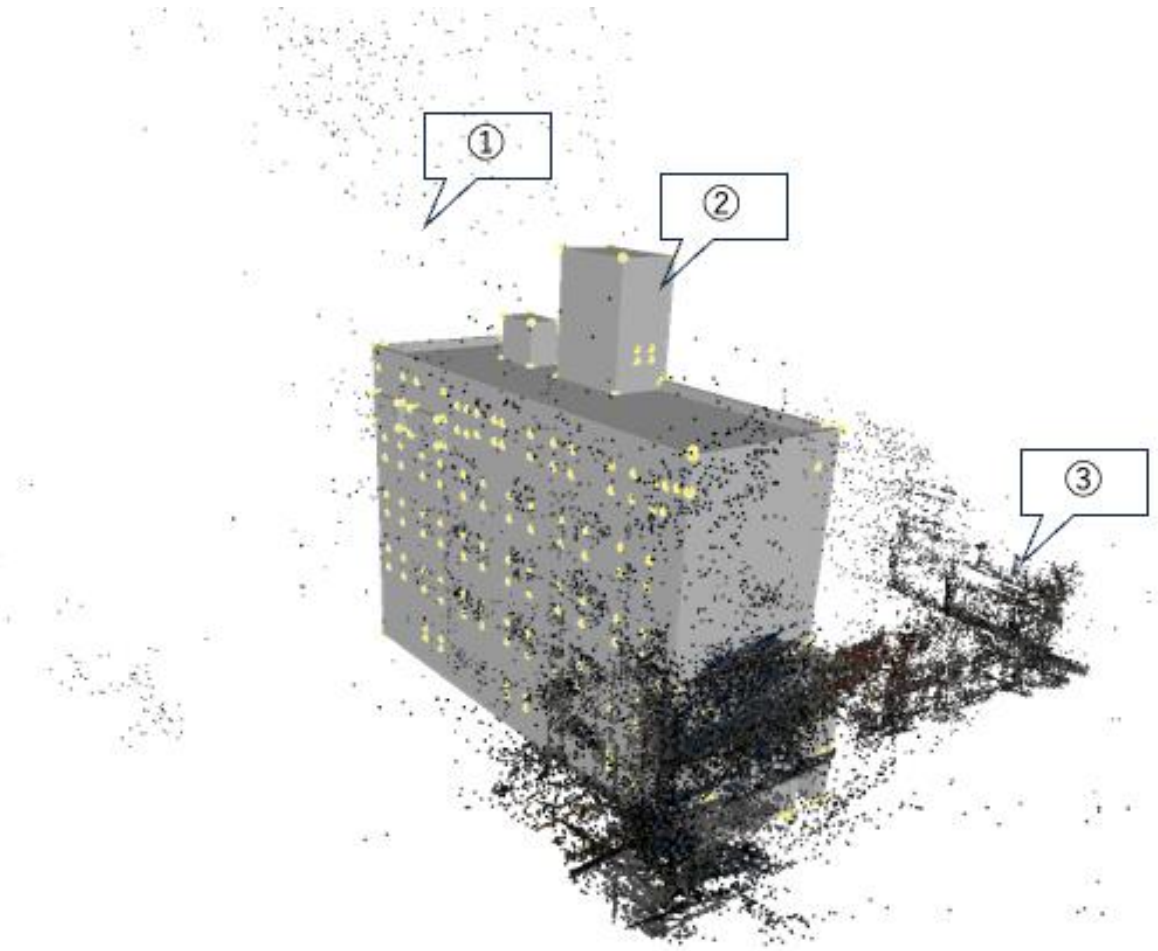


図 8-18 点群 to 点群対応化におけるデータノイズ

2) 検証結果詳細

- 検証エリア１：ケーキショップ Dolce 周辺

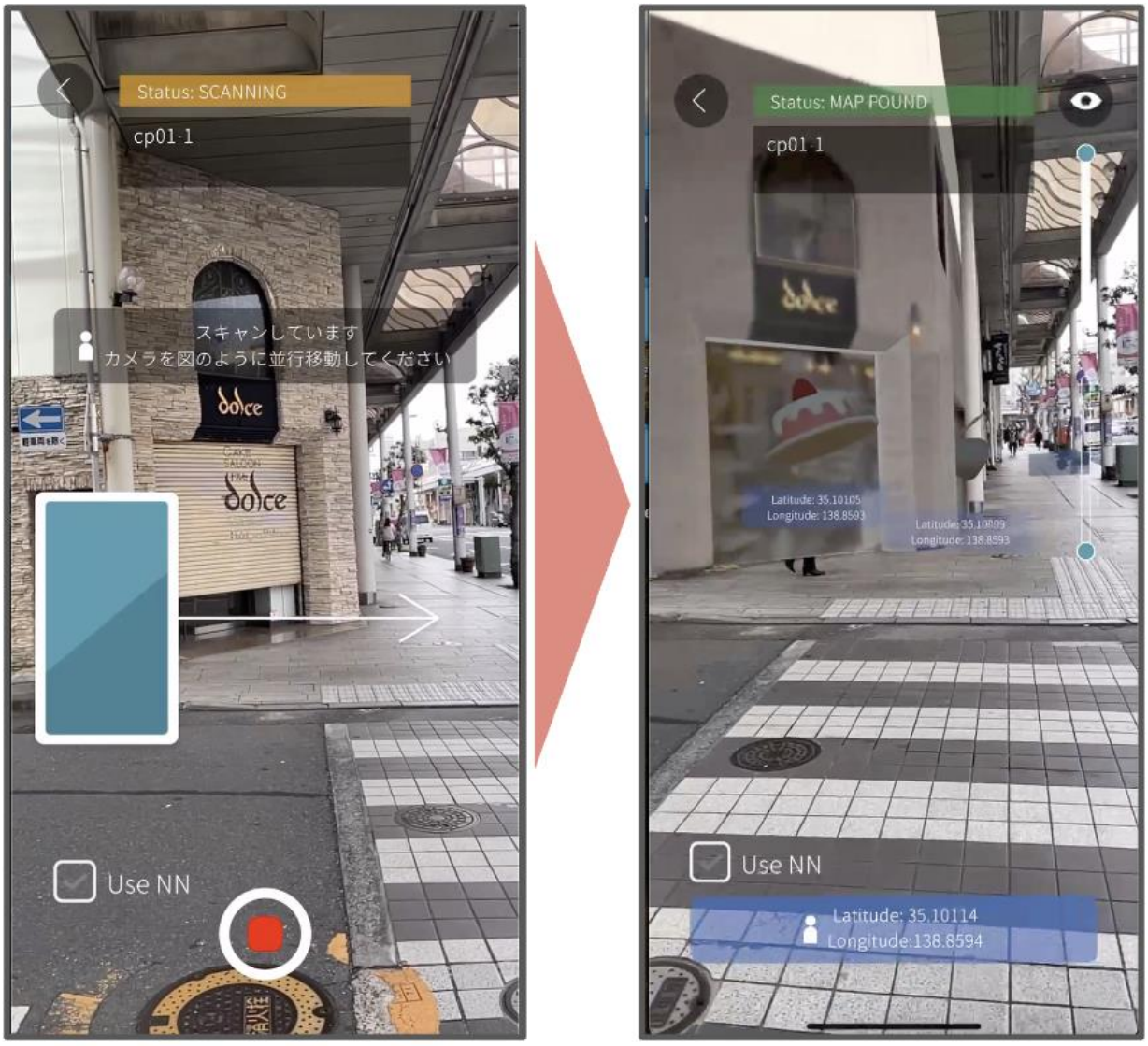


図 8-19 検証エリア１における検証結果

- 発生事象
  - 期待どおり、高精度で自己位置推定できた
  - 難易度が高いと予測していたガラスの多い面も、壁面のエアコン室外機等をポイントに自己位置推定できた
  - 店舗のシャッター開閉に関わらず自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 2：くわはら矯正歯科医院周辺

図 8-20 検証エリア 2 における検証結果

- 発生事象
  - 期待どおり、高精度で自己位置推定できた
  - 店舗のシャッター開閉に関わらず自己位置推定できた
  - 車庫内の車の有無に関わらず自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア３：魚ぶん周辺

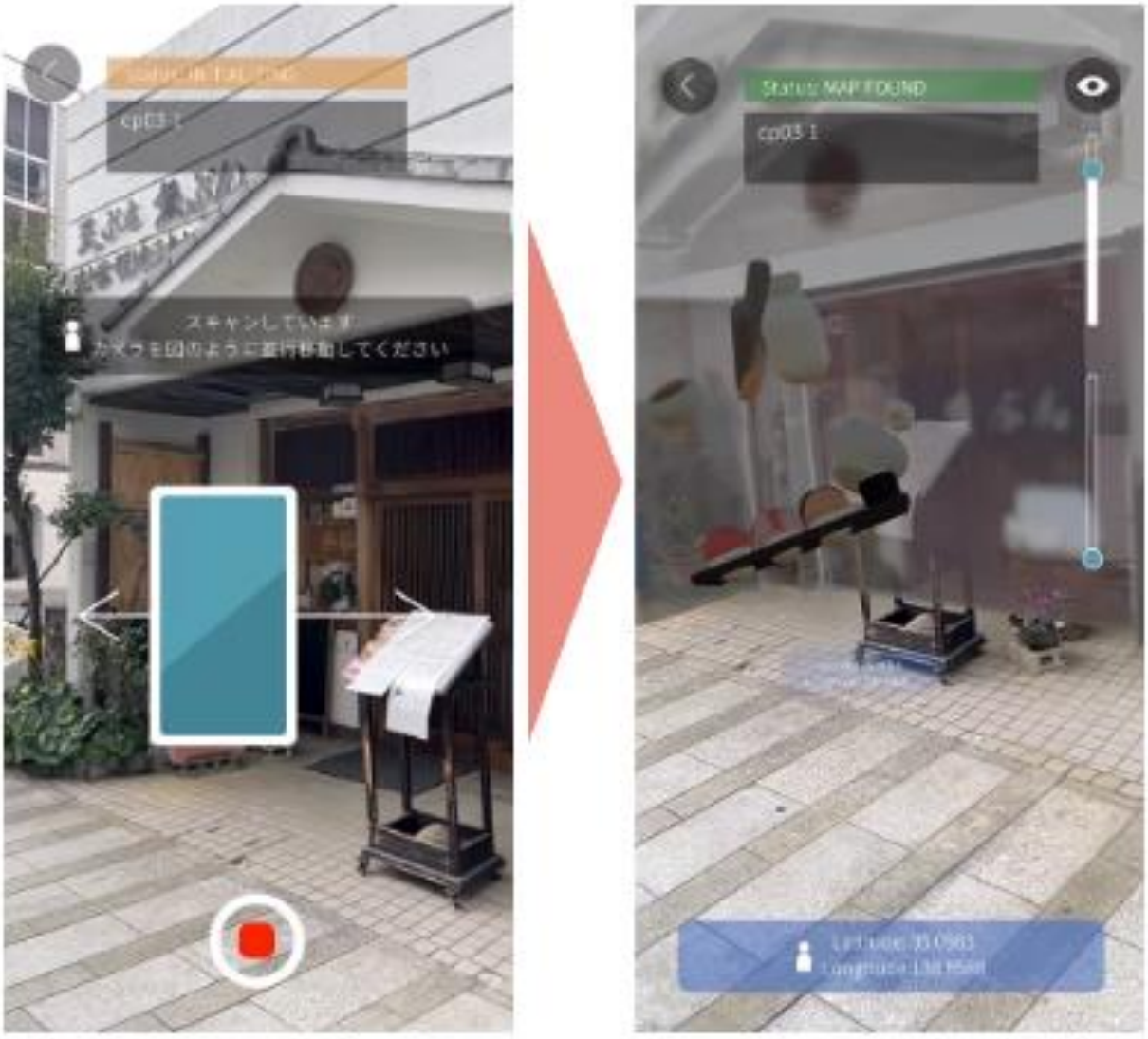


図 8-21 検証エリア３における検証結果

- 発生事象
  - 期待どおり、高精度で自己位置推定できた
  - 看板の位置のズレなどの影響もなく自己位置推定できた
  - 店舗の営業有無（看板や暖簾の有無）に関係なく自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 4 ：デザートレストラン Grandma 上土本店周辺

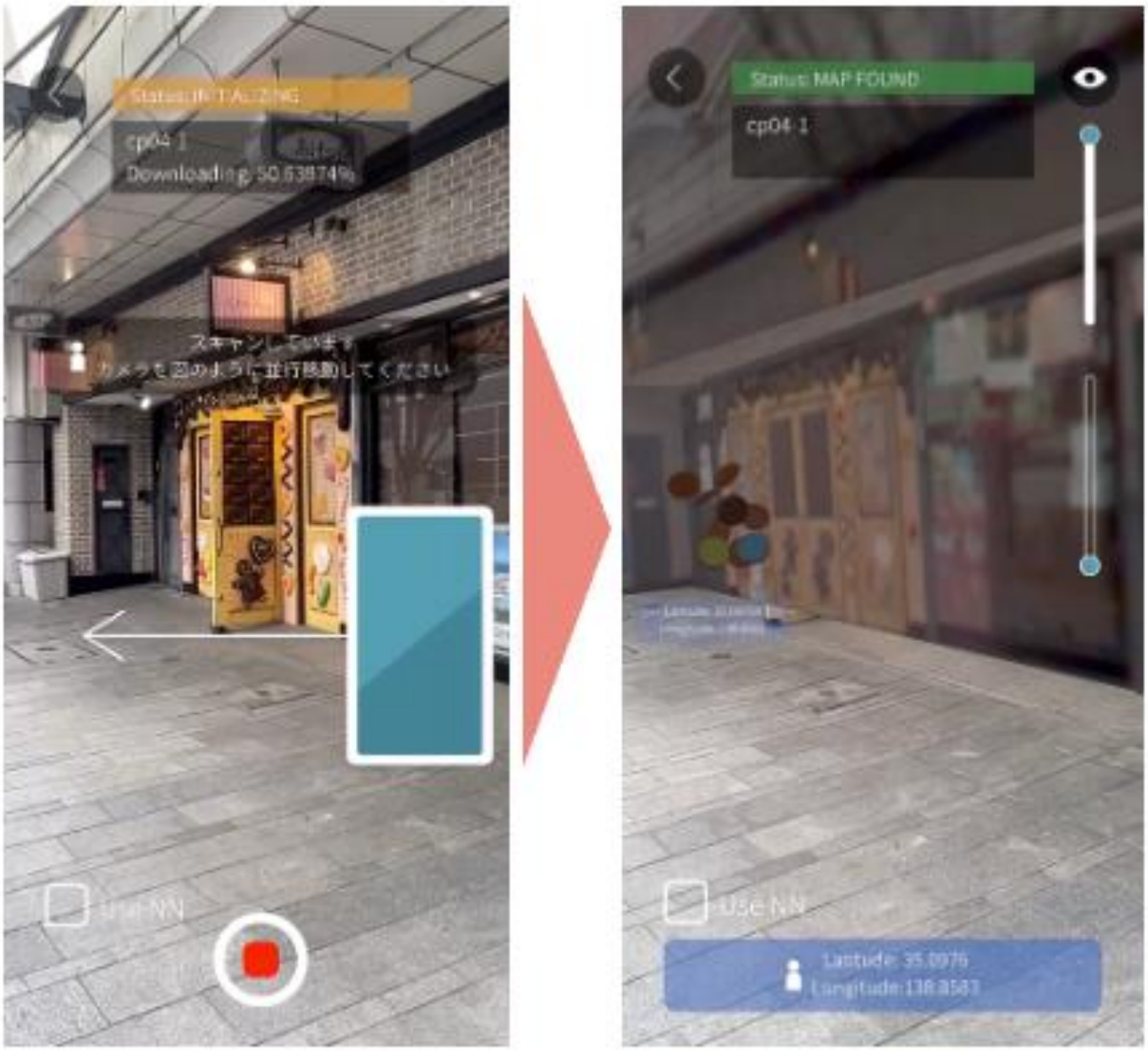


図 8-22 検証エリア 4 における検証結果

- 発生事象
  - 期待どおり、高精度で自己位置推定できた
  - 扉の開閉に関わらず自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 5 ：美容室　GUK 周辺

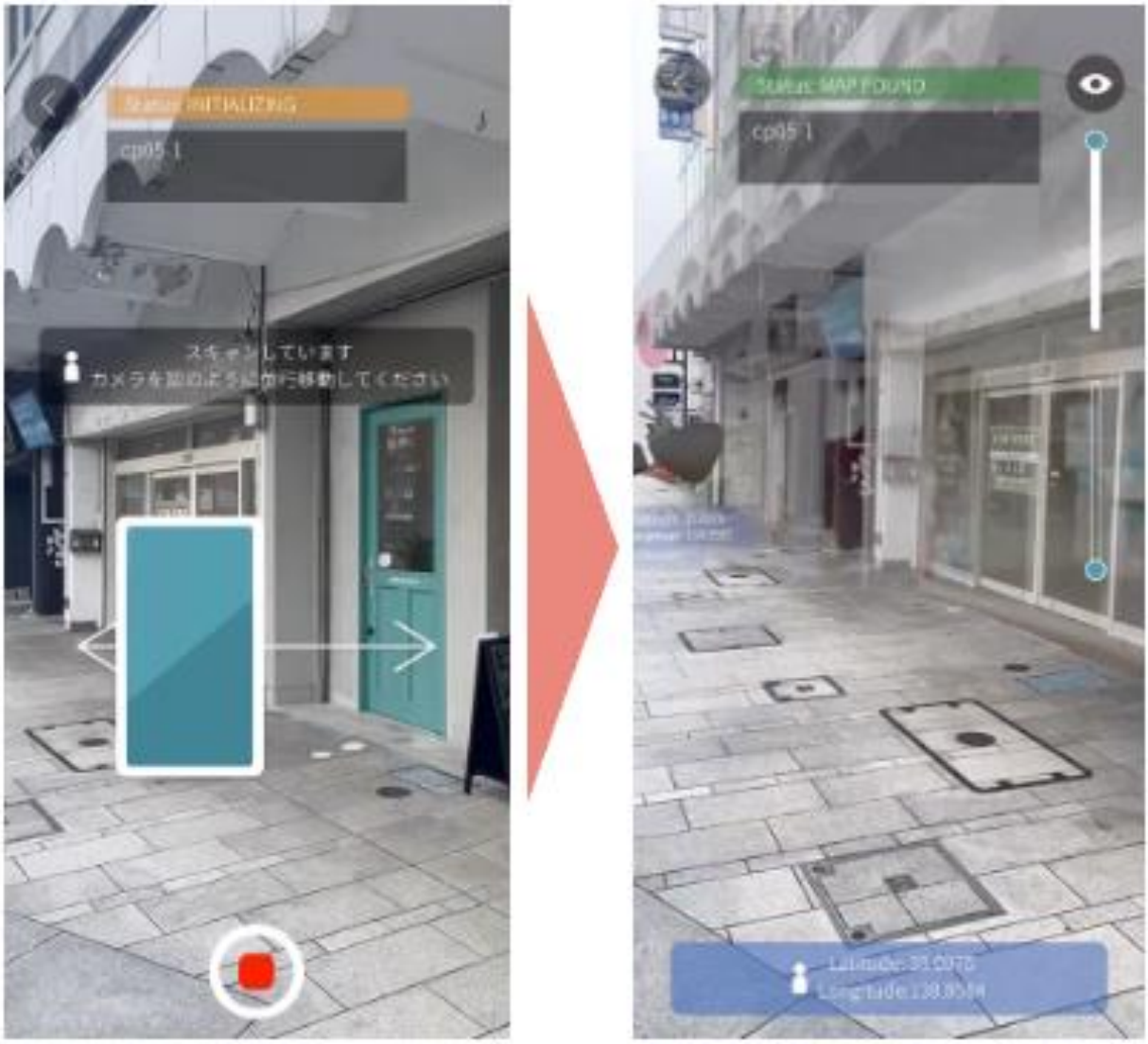


図 8-23 検証エリア 5 における検証結果

- 発生事象
  - 壁面に大きな特徴はなかったものの、看板や窓枠などをポイントに比較的高精度に自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 6 ：サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺

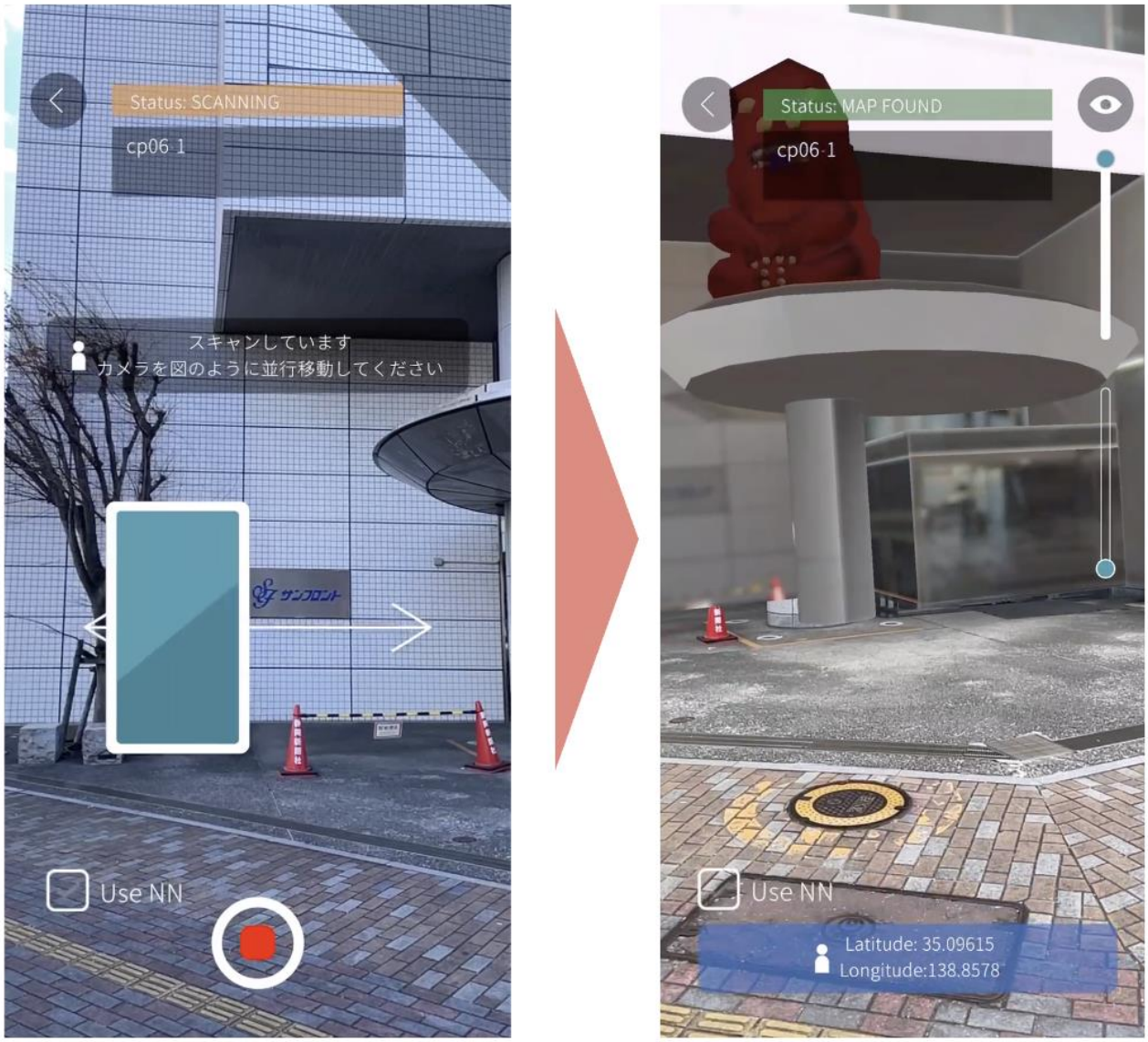


図 8-24 検証エリア 6 における検証結果

- 発生事象
  - 自己位置推定は実施できたが、当初の想定どおり、外壁を長方形のシンプルなパターンで覆われているため高精度での自己位置推定には至らなかった
- 課題
  - 自己位置推定の結果表示した 3D コンテンツの位置精度が低かった
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - VPS が苦手な「同一パターンが連続するエリア」を避けて自己位置推定を実施するよう、アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 7 ：（有）一樹商事周辺

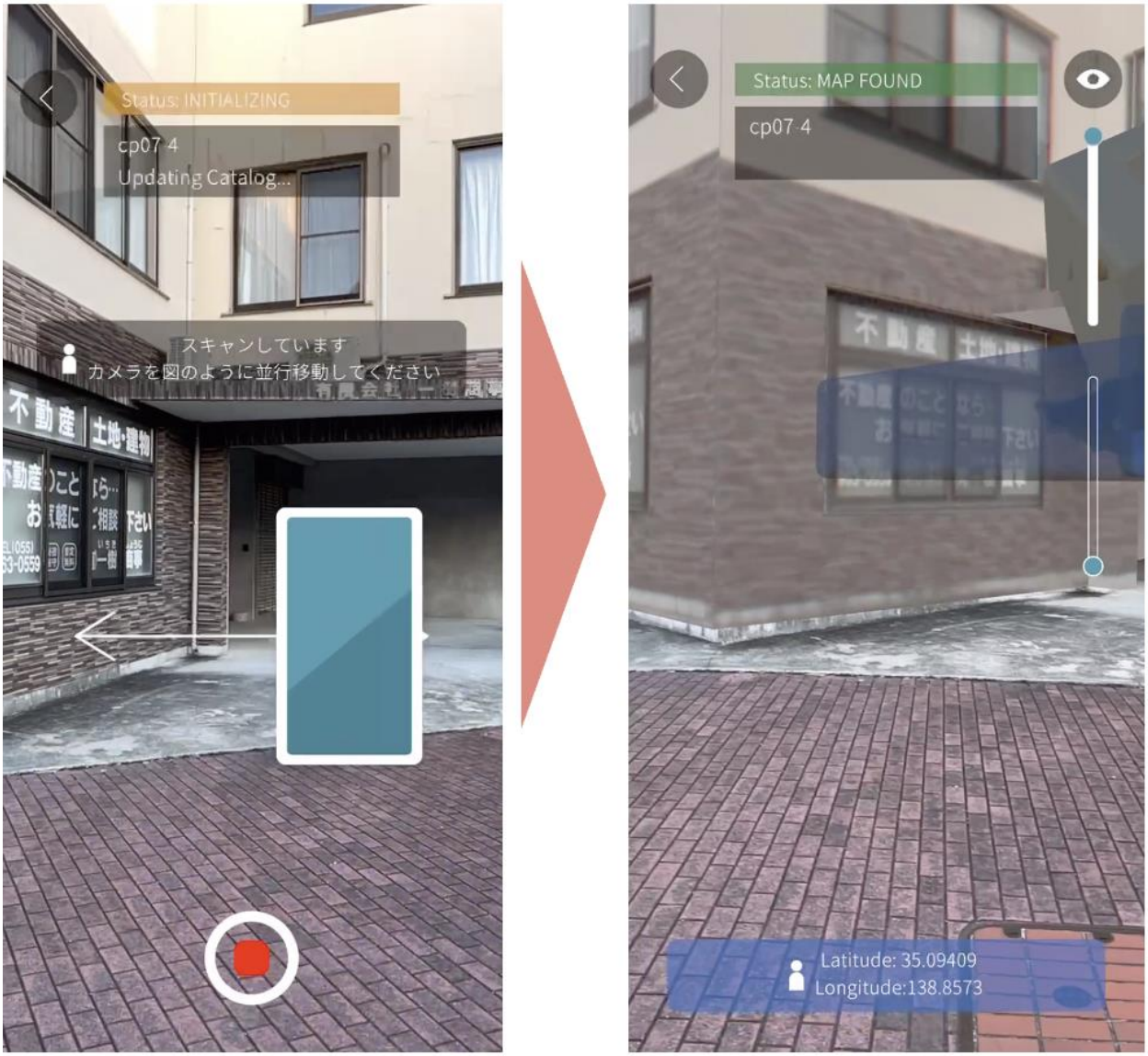


図 8-25 検証エリア 7 における検証結果

- 発生事象
  - 期待どおり、高精度で自己位置推定できた
  - 窓内の文字等を特徴点として捉えることを期待していたが、窓枠などを優先的に特徴として捉えることで自己位置推定を実現していた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア８：不動院周辺

図 8-26 検証エリア８における検証結果

- 発生事象
  - 道路から建物まで若干距離があり、特徴を掴みにくくやや難易度の高いと想定していたが、建物の梁などを中心に特徴点を抽出し高精度で自己位置推定できた
- 課題
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 9 ：ファミリーマート 沼津下河原町店周辺

図 8-27 検証エリア 9 における検証結果

- 発生事象
  - 自己位置推定処理は成功したものの、測定不能な精度誤差が発生した
- 課題
  - 測定不能な精度誤差が発生した
- 課題への対応策
  - 当初の想定どおりではあるものの、広い駐車場に囲まれており道路から建物までの距離がある、かつ周辺の建物と距離もあるため、共通の特徴が掴めず適切に自己位置推定できなかったと考える
  - VPS が苦手な「環境変化の大きなエリア」を避けて自己位置推定を実施するよう、アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

- 検証エリア 10：（有）するが水産周辺

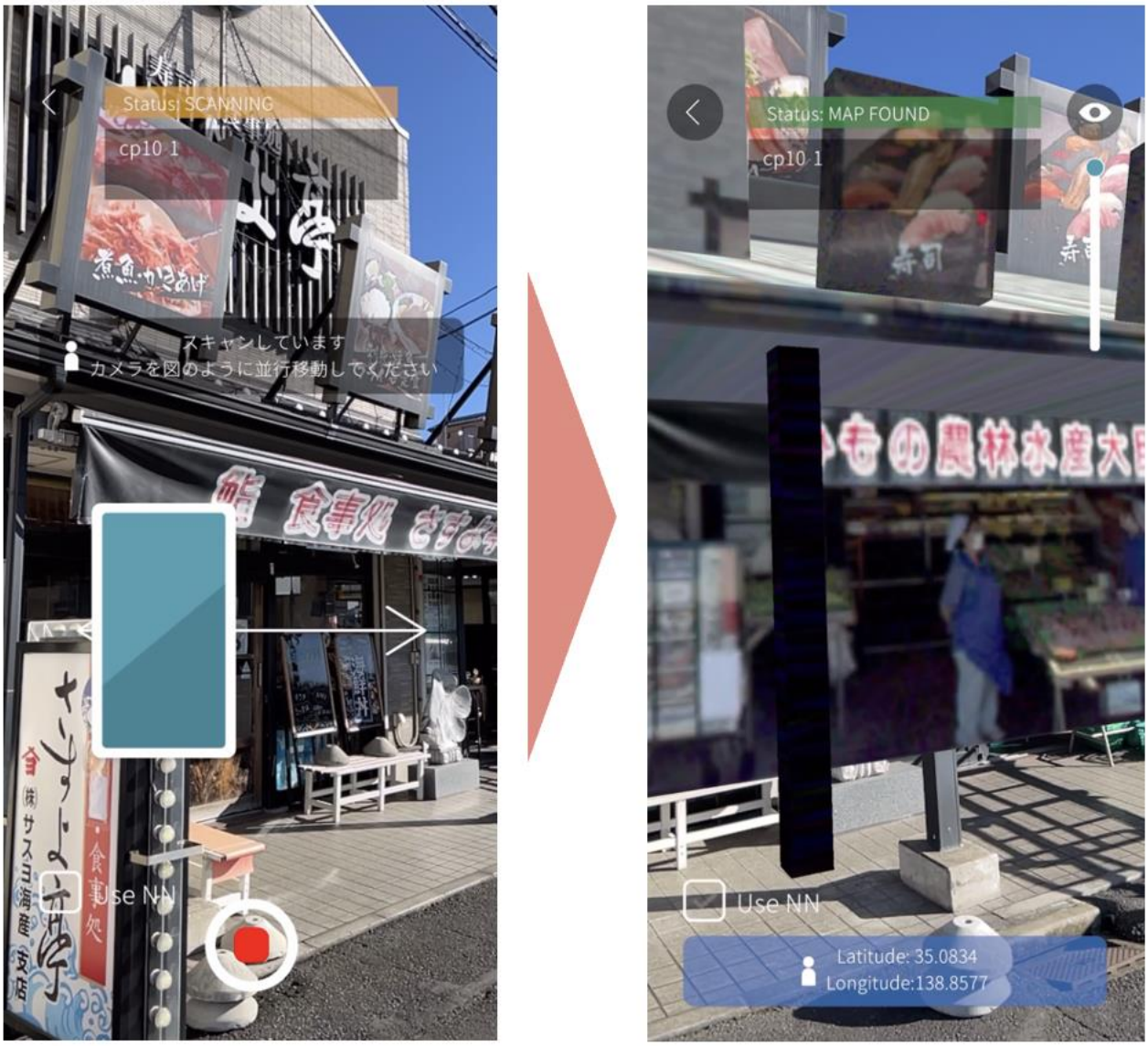


図 8-28 検証エリア１０における検証結果

- 発生事象
  - 人通りも多く環境変化が大きかったため、自己位置推定は期待よりは低い精度となったが、目標値はクリアできた
- 課題
  - 自己位置推定の精度は期待より低かった
  - カメラの向きや位置によっては自己位置推定に時間がかかるケースがあった
- 課題への対応策
  - VPS が苦手な「環境変化の大きなエリア」を避けて自己位置推定を実施するよう、アプリ上でカメラの向きや位置を誘導する仕組みを検討することが考えられる

9. スマートフォン向けアプリケーション：BtoB ビジネスでの有用性検証

9-1. 検証目的

実証仮説に基づき、以下の検証目的を設定する。

【検証仮説】
3D 都市モデルを活用した AR ナビゲーションを開発することにより、既存の地図アプリやナビに対してより直感的な案内や、個人の位置と趣味嗜好に合わせたより詳細なルート案内が可能となる。
将来的には、これまで GPS では実現できなかった建物内でのナビゲーションやラストワンマイルモビリティのより柔軟かつ正確なコントロールにも期待ができる。

本検証では、下記 2 点について、BtoB ビジネスに向けた有用性検証を行った。

- システムの精度・性能検証
  - 今回の実証実験では既存の VPS 技術を指標に技術検証を実施した。
  - 一方、実際の利用用途やユーザーによって、求める技術の品質は異なる。
  - 今年度開発した本システムにおける自己位置推定の技術的な十分性と期待値について技術者ではない者に確認する
- AR ナビゲーションの需要検証
  - AR ナビゲーションを想定した簡易アプリケーションを開発することで、既存の地図アプリやナビと比較した際の優位性を確認する

9-2. 検証方法

検証方法としては、被験者に対してデモンストレーションを取り入れたアンケートを実施した。
（アンケートの項目については「9-4 ヒアリング・アンケートの詳細」で記載）

事業者向けヒアリングの実施方法
- 会場：沼津駅～沼津港
- 機材：デモ用に以下のスペックのスマートフォンを用意する。
  - iPhone SE
  - CPU：
  - 通信環境：4G 回線 SIM

9-3. 被験者

今回の実証実験では、プロジェクト関係者及び非関係者の技術者でない者をターゲットとしている。
これらのユーザーとして該当する以下の方々にデモ及びアンケートを行い、本システムの価値を検証する。

表 9-1 被験者リスト

分類	具体名称	部署	役職	人数
事業者/ ユーザー	TOPPAN 株式会社	情報コミュニケーション事業本部 ソーシャルイノベーション事業部 公共事業推進センター アカウントプロデュース本部	-	1
		情報コミュニケーション事業本部 ソーシャルデザイン事業部ソーシャルイノベーションセンター市場開発本部	担当課長	1
	ホロラボ	-	代表取締役 CEO	1
	沼津市役所	まちづくり政策課　交通政策室	主任	1
			主事	1

9-4. ヒアリング・アンケートの詳細

9-4-1. アジェンダ・タイムテーブル

スマートフォン向け AR アプリケーションとして今回の実証実験で開発した VPS システムを搭載し、本システムがコンシューマ向けコンテンツの利用に対して必要十分な位置精度及び処理性能（時間）で活用できるかを検証する。
具体的には今回の実証実験の目的を共有後、実際に AR アプリケーションを実体験いただいた上で、その利用結果や操作性に対する意見をアンケート形式で収集する。

表 9-2 アジェンダ・タイムテーブル

No.	アジェンダ	所要時間
1	実験の目的を説明	10 分
2	作業手順のデモ、説明（既存と新規開発）	10 分
3	操作の体験（本年度ユースケースで開発したシステム）	60 分
4	アンケート回答（Google フォームでの回答）	10 分

9-4-2. アジェンダの詳細

表 9-3 アジェンダの詳細

No	アジェンダ（再掲）	内容
1	実験の目的を説明	● 今回の実証実験でアプローチする課題や背景の説明 ● 今回の実証実験の比較対象となる従来技術の説明 ● 今回の実証実験で用いるシステムの提供価値 ● システムの全体像の説明
2	作業手順のデモ、説明 （既存と新規開発）	● 新規開発システム ➢ iPhone SE 上で、本年度ユースケースで開発したシステムを用いた 3D 都市モデルを利用した AR アプリケーションのデモ、操作説明
3	操作の体験	● 上記デモ内容を事業者が体験
4	アンケート回答	● Google フォームでアンケート回答を依頼し、その場で回答（10 分）、送信を依頼

9-4-3. 検証項目と評価方法

開発した VPS の精度、性能及び AR アプリケーションのニーズ確認を検証項目とし、それぞれに評価した。

表 9-4 検証項目と評価方法

検証観点	No	検証項目	評価
VPS の精度	1	3D のアイコン表示位置は適切か	● 選択肢は「不適切」を 1、「適切」を 5 とした 5 段階で設定
	2	精度の重要性	● ラジオボタンで下記 4 つから選択 ➢ 非常に重要（表示誤差 30cm 未満の非常に高い精度が必要） ➢ 重要（表示誤差 1m 未満の高い精度が必要） ➢ やや重要（表示誤差 3m 未満の中程度の精度が必要） ➢ あまり重要ではない（表示誤差 3m 以上でも、なんとなく近くに何があるかわかれば良い）
	3	その他精度についての気づき	● アンケートで自由記入欄を設定
VPS の性能	4	処理性能は適切か	選択肢は「不適切」を 1、「適切」を 5 とした 5 段階で設定。
	5	処理性能の重要性	● ラジオボタンで下記 4 つから選択 ➢ 非常に重要（アプリ起動後 3 秒以内で即座に表示してほしい） ➢ 重要（アプリ起動後 10 秒以内で表示してほしい） ➢ やや重要（早いに越したことはないが、30 秒～1 分程度は許容できる） ➢ あまり重要ではない（興味のあるアクティビティのレコメンドなど、価値を提供してくれるなら数分程度の時間は気にしない）
	6	その他処理性能についての気づき	アンケートで自由記入欄を設定

AR ナビの有用性	7	AR ナビが実現した場合使ってみたいか	選択肢は「そう思わない」を 1、「そう思う」を 5 とした 5 段階で設定。
全体	8	実証全体へのフィードバック	アンケートで自由記入欄を設定

9-4-4. 実証実験の様子

図 9-1 実証の様子 1（AR アプリから見た様子）

図 9-2 実証の様子 2（参加者が AR コンテンツの表示を確認している様子）

図 9-3 実証の様子 3 (AR アプリから見た様子 – ケーキ屋を表示)

図 9-4 実証の様子 4 (AR アプリから見た様子 – 画面上にルート案内を表示)

図 9-5 実証の様子 5. (AR アプリから見た様子 – 魚市場へ到着)

図 9-6 実証の様子 6 (AR アプリの利用方法の説明①)

図 9-7 実証の様子 7.（AR アプリの利用方法の説明②）

9-5. 検証結果

3D 都市モデルを活用した VPS 機能を搭載したスマートフォン向けアプリケーションを提供し、AR ナビゲーションに対して市場に一定の期待や価値があることが認められた。また、ユーザーの視点において、精度や処理性能への期待値としては、精度としての誤差 1m 以内、性能としてコンテンツ表示までの時間 10 秒以内であり、本プロジェクトにおける KPI が適切であったことが確認できた。一方、精度・性能ともに全てのユーザーが満足するものにはなっておらず、特に性能においては、使い方やタイミングによってコンテンツ表示までに時間がかかり、ユーザーにストレスを与えるケースがあった。

Q1 開発した AR アプリにおける位置の精度は良いか。また、AR コンテンツが表示されるまでの時間は良いか

位置精度については全参加者のうち全員（5/5 名）が「非常に良い」か「良い」と回答。精度が悪いと感じる参加者はおらず、ナビゲーションとして十分活用できる精度が実現できていることが明らかになった。
処理性能についても 60%以上（3/5 名）が「非常に良い」か「良い」と回答したものの、一部「悪い」と感じる参加者もいた。これは、カメラを向ける位置や人通りなどの周囲のノイズに依存して AR コンテンツを表示するまでに時間がかかることがあったためと考える。これはカメラの向け方や利用方法で改善できる部分ではあるものの、一般ユーザーの利用を想定する場合、ユーザーが意識することなく最適な利用を促せる必要がある。このことから、将来的な活用に向けてはシステムの処理を高速化することはもちろん、アプリ内でのガイド表示など UI/UX の観点も併せてシステム全体を通してユーザーが「処理が遅い」と感じないための仕掛けを検討する必要があることが明らかになった。

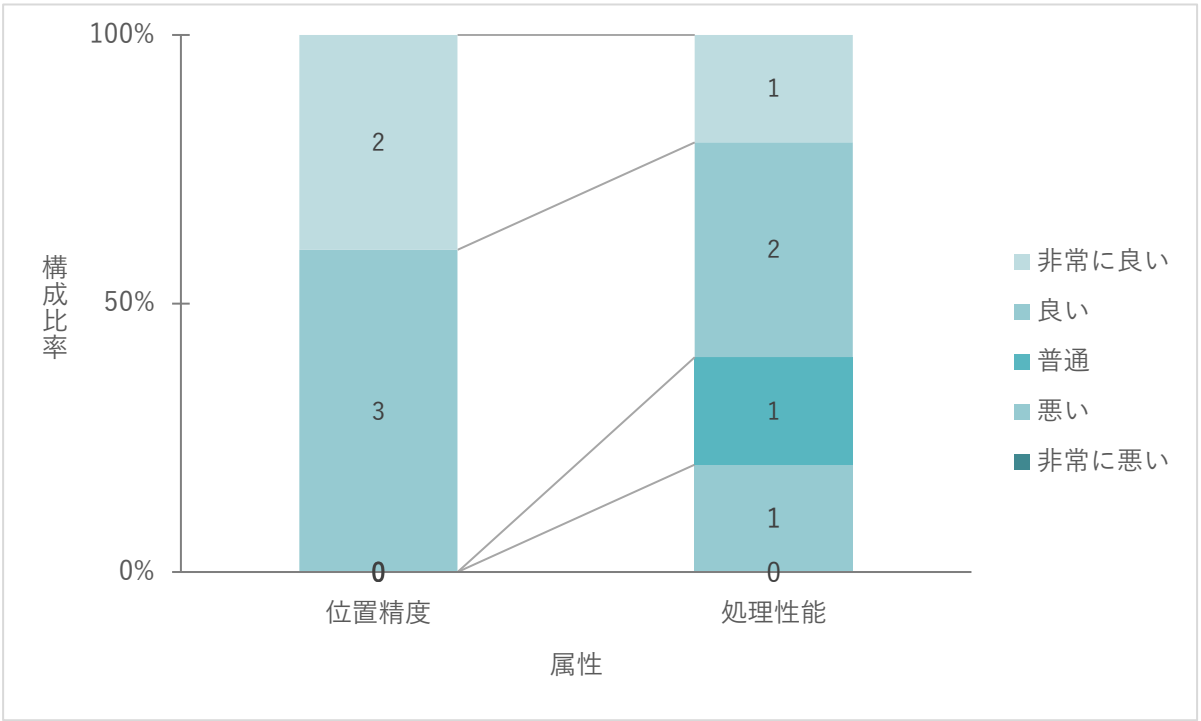


図 9-8 アンケート結果-開発した AR アプリに対するユーザーの評価

Q2 AR ナビゲーションの利用に向けて、コンテンツの表示位置精度や処理性能はどの程度重要か

位置精度、処理性能ともにほぼ全ての参加者が「重要」もしくは「非常に重要」と回答した。特に処理性能は既存のアプリケーションでの操作感など、ユーザーが比較するための基準を持っているためより重要視されているように見える。実際、定性的なコメントでは精度よりも性能に対するコメントが多く、中には「ポケモン Go」と対比してコメントを残した参加者もいた。

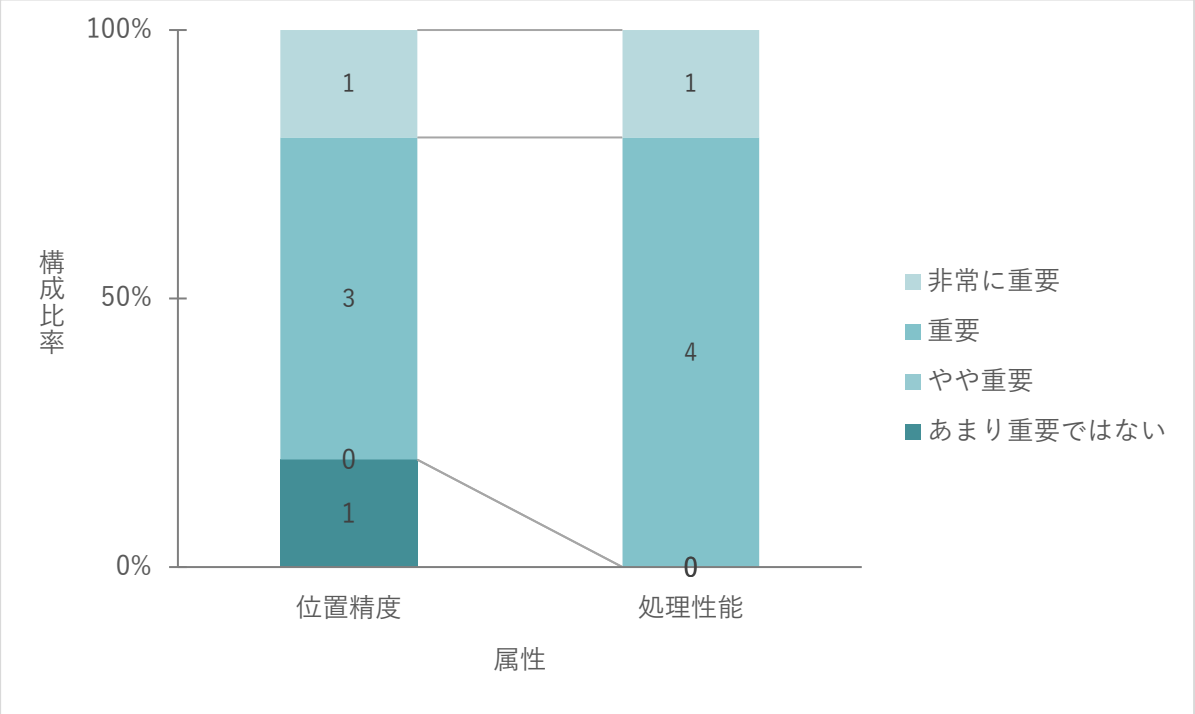


図 9-9 アンケート結果-開発した AR アプリに対する精度や性能の重要性

表 9-5 関連する定性コメント

No	定量調査の結果・示唆	関連する定性コメント
1	コンテンツの表示位置精度は十分なものとなっている。	● 多少の誤差はあれど、およそ正しく表示されていたかと思う。 ● 階段の上り下りなど、ある程度移動すると誤差が大きくなるケースがあった。
2	処理性能を重要視する参加者が多く、操作性の向上も併せた改善が必要。	● 表示されない時に、リロードやリトライの指示だしがあるとよい。 ● 複数回スキャニングが必要なのであれば、1 回あたりにかかる時間は 1 0 秒以内が望ましいと思う。 ● 今回の内容は非常にポケモン Go と近しいものと感じました。ただ、それが故に処理時間についてもポケモン Go と同じ位（用途が違うた

		め 10 秒から 20 秒位が許容範囲かと…）は求められてしまうのかなと思います。おそらく 30 秒以上かかるとユーザーはこのアプリケーション活用を諦めるのではないかと思う。

Q3 将来 AR ナビゲーションを使ってみたいと思うか

全ての参加者が「是非使いたい」もしくは「使いたい」と回答した。このことから、一般ユーザーに利用してもらうためには精度・性能・使いやすさなど多くの課題があるものの、AR ナビゲーションが実現した世界のイメージを与え、将来への期待を持たせることができたと言える。

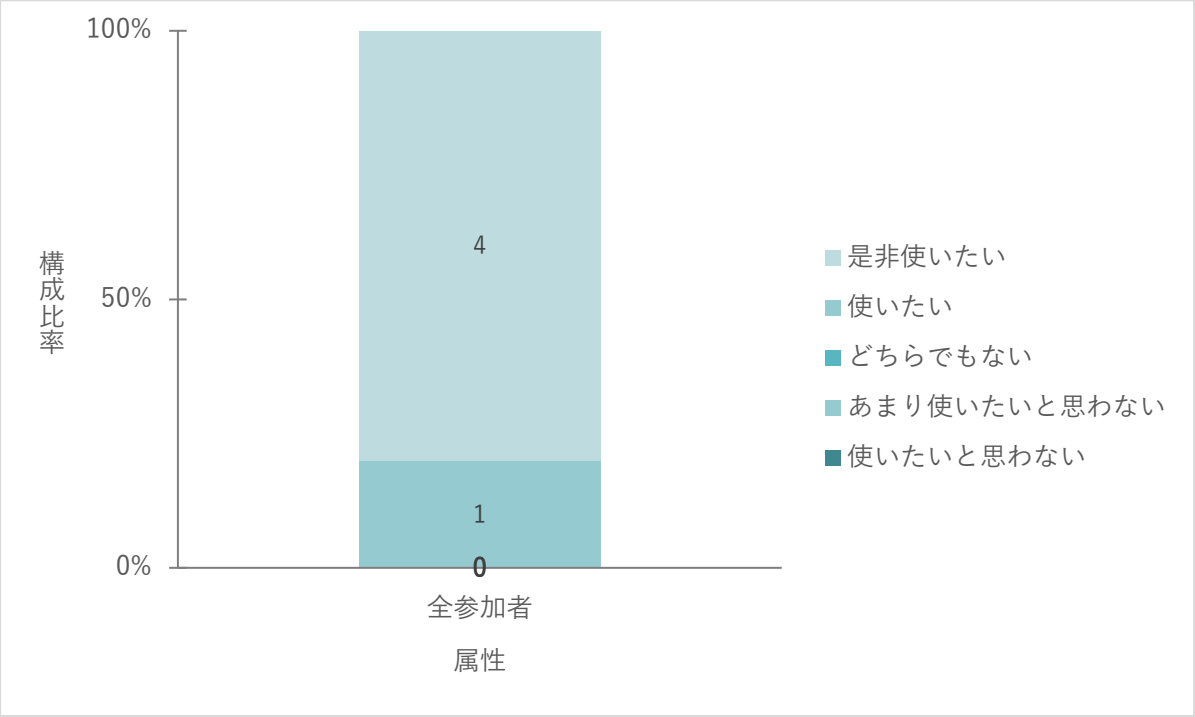


図 9-10 アンケート結果-AR ナビゲーションへの期待

10. 両実証の比較

10-1. VPS 精度の検証

車両向けシステムは車両での自動運転を想定し、広域を対象とした vSLAM 技術をベースとしたリアルタイム性を重視したシステムであるのに対し、スマートフォン向けシステムは歩行者やラストワンマイルモビリティなど停止を許容される状況下での利用を想定し、SfM 技術をベースとした精度を重視するシステムとなっている。さらに車両向けでは 3D 都市モデルを VPS マップとして利用し、3D 都市モデル対実写画像のマッチングを試みているが、スマートフォン向けシステムは実写画像から作成したマップと 3D 都市モデルを位置合わせして利用している。そのため、精度のみで双方の技術を比較・評価することは難しいが、参考のため、今回の実証実験における 2 つの VPS 技術（車両向け・スマートフォン）について VPS の位置精度誤差を比較した。

10-1-1. 比較の方法

実証エリア内に比較用の地点を設け、その周辺で両方の VPS を実際に稼働させ自己位置推定の挙動、精度を確認することとした。比較用の地点はスマートフォン向けシステムが対象としている地点が実証エリア内で 10 か所あり、それをそのまま利用することとした。
車両向けシステムでは、スマートフォン向けシステムの条件と合わせるため、比較用地点周辺にてステレオカメラを手持ちで移動し撮影した動画データをシステムに入力し自己位置推定された座標に対して Rviz 上で実際の位置からの距離を計測しそれを精度として用いた。スマートフォン向けシステムでは 8-2-3 で記載した精度検証により得た数値を使用する。

10-1-2. 比較結果

結果として、車両向けシステムにおいては自己位置推定できない場所が多く、遠方からでも目立つコントラストのはっきりした対象物に対してのみ自己位置推定が可能な結果となった。その場合においても精度は 1m より誤差が大きかった。
一方でスマートフォン向けシステムでは一部の場所を除いて比較的高精度な自己位置推定が可能な結果となったが、NO.6（サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺）のようなシンプルな建物がパターン的に連続する建築物の付近や、NO.9（ファミリーマート 沼津下河原町店周辺）のような計測値から特徴点となる地物までの距離が比較的遠い場所では大きな誤差が発生することがわかった。
車両向けシステムでは、3D 都市モデルからマップを作成しているため、3D 都市モデルで表現されていない細かい局所的な特徴ではなく、遠方からのはっきりした特徴を利用して自己位置推定を行っているため、スマートフォン向けシステムと同条件で比べた場合、より精度が悪くなっていると考えられる。（例えば NO.9 の特徴点が遠方に多く分布している地点では結果が逆転している。）
VPS の各システムの特性の違いは大きく、使用用途に合わせたシステムを使うことや、用途に合わせたシステム構成、チューニングが重要である。

表 10-1 両実証の精度比較

No.	検証エリア	VPS の精度誤差 ※5 ㎝単位での計測		示唆
		車両向け システム	スマートフォン 向けシステム	
1	ケーキショップ Dolce 周辺	測定不能	60	● スマートフォン向けシステムは高精度な自己位置推定が可能だった ● ガラス面の多い建築物も外壁のエアコン室外機を検出のポイントとしていた ● 店舗のシャッターが閉まっており、3D 都市モデルのテクスチャとは外観が異なっていたが問題なく自己位置推定した
2	くわはら矯正歯科医院周辺	200	20	● 車両向け、スマートフォン向け双方で自己位置推定が可能だった ● 1 と同様にシャッターが閉まっており、また車庫内の車の有無にかかわらず問題なく自己位置推定した
3	魚ぶん周辺	測定不能	20	● スマートフォン向けシステムは高精度な自己位置推定が可能だった ● 実証実施日に店舗が休業日のため、

				看板や暖簾が無い状態で実施したが、問題なく自己位置推定した
4	デザートレストラン Grandma 上土本店周辺	測定不能	35	● スマートフォン向けシステムは高精度な自己位置推定が可能だった ● 実証中に扉の開閉があったが、問題なく自己位置推定した
5	美容室　GUK 周辺	測定不能	30	● スマートフォン向けシステムは高精度な自己位置推定が可能だった ● 壁面に大きな特徴はなかったものの、看板や窓枠などをポイントに比較的高精度に自己位置推定した
6	サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺	300	160	● 車両向け、スマートフォン向け双方で自己位置推定が可能だった ● 外壁を長方形のシンプルなパターンで覆われている建築物であり特徴点を判定しづらいため、双方の VPS で高精度での自己位置推定には至らなかった
7	（有）一樹商事周辺	500 以上	40	● 車両向け、スマートフォン向け双方で自己位置推定が可能だった ● 車両向けシステムでは移動しながら特徴点を絞り込むことが出来ず大きな位置ズレが発生した ● スマートフォン向けでは窓枠やエアコンの室外機を特徴点として捉えることで高精度な自己位置推定が可能だった
8	不動院周辺	500 以上	40	● 車両向け、スマートフォン向け双方で自己位置推定が可能だった ● 車両向けシステムでは移動しながら特徴点を絞り込むことが出来ず大きな位置ズレが発生した ● スマートフォン向けシステムでは建物の梁を特徴点として捉えることで高精度な自己位置推定が可能だった
9	ファミリーマート沼津下河原町店周辺	300	測定不能	● 車両向けシステムは観測地点前後の特徴及び、遠方の特徴点も含めて自己位置推定が可能だった

				● スマートフォン向けシステムでは、観測地点が広い駐車場に囲まれており道路から建物までの距離がある、かつ周辺の建物と距離もあるため、共通の特徴が掴めず適切に自己位置推定することができなかった
10	（有）するが水産周辺	100	70	● 車両向け、スマートフォン向け双方で自己位置推定が可能だった ● 双方とも看板を特徴点として捉えており、人通りがある状況でも比較的高精度に自己位置推定することができた

10-1-3. 詳細結果

1) ケーキショップ Dolce 周辺

- 車両向けシステム：測定不能
- スマートフォン向けシステム：誤差 60 ㎝
  - 難易度が高いと予測していたガラスの多い面も、壁面のエアコン室外機等をポイントに自己位置推定できた
  - 店舗のシャッター開閉に関わらず自己位置推定できた

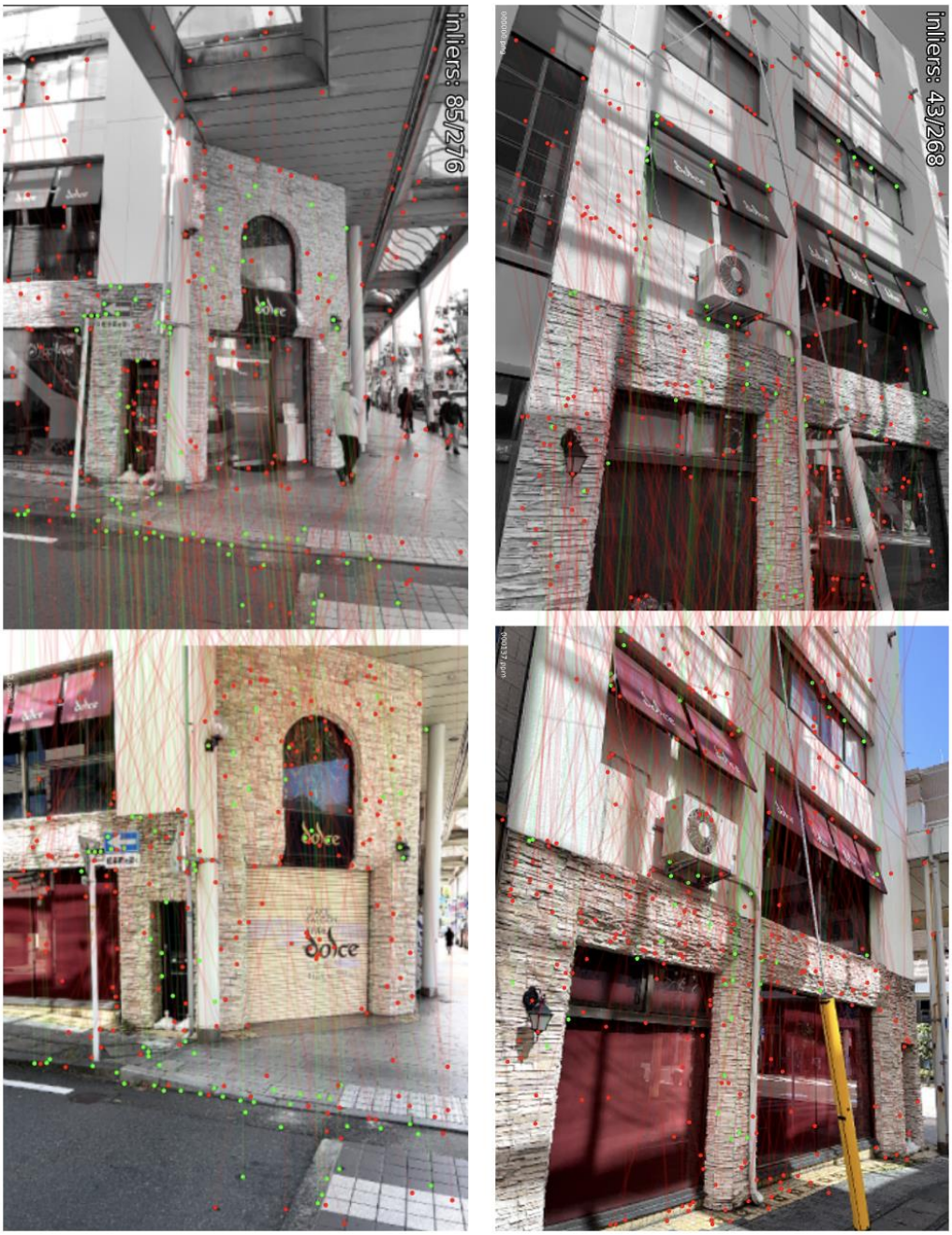


図 10-1 ケーキショップ Dolce 周辺での自己位置推定

2) くわはら矯正歯科医院周辺

- 車両向けシステム：誤差 200 ㎝
  - 誤差は大きいが自己位置推定できたが、不安定であった

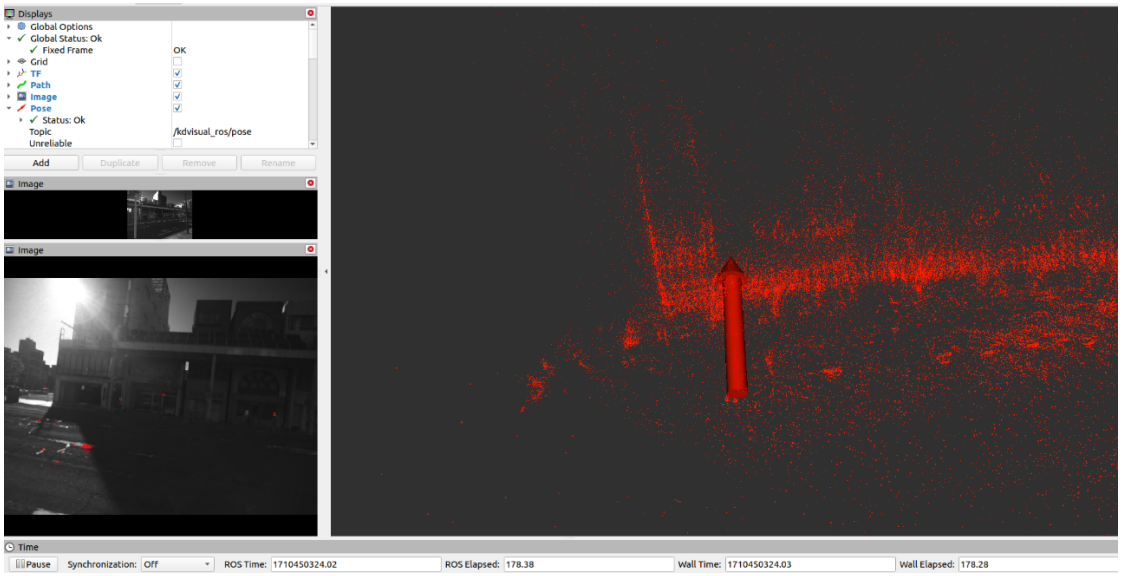


図 10-2 くわはら矯正歯科医院周辺での自己位置推定

- スマートフォン向けシステム：誤差 20 ㎝
  - シャッターの開閉、また車庫内の車の有無に関わらず自己位置推定できた

図 10-3 くわはら矯正歯科医院周辺での自己位置推定

3) 魚ぶん周辺

- 車両向けシステム：測定不能
- スマートフォン向けシステム：誤差 20 ㎝
  - 店舗の営業状況による看板や暖簾の有無に関係なく自己位置推定できた

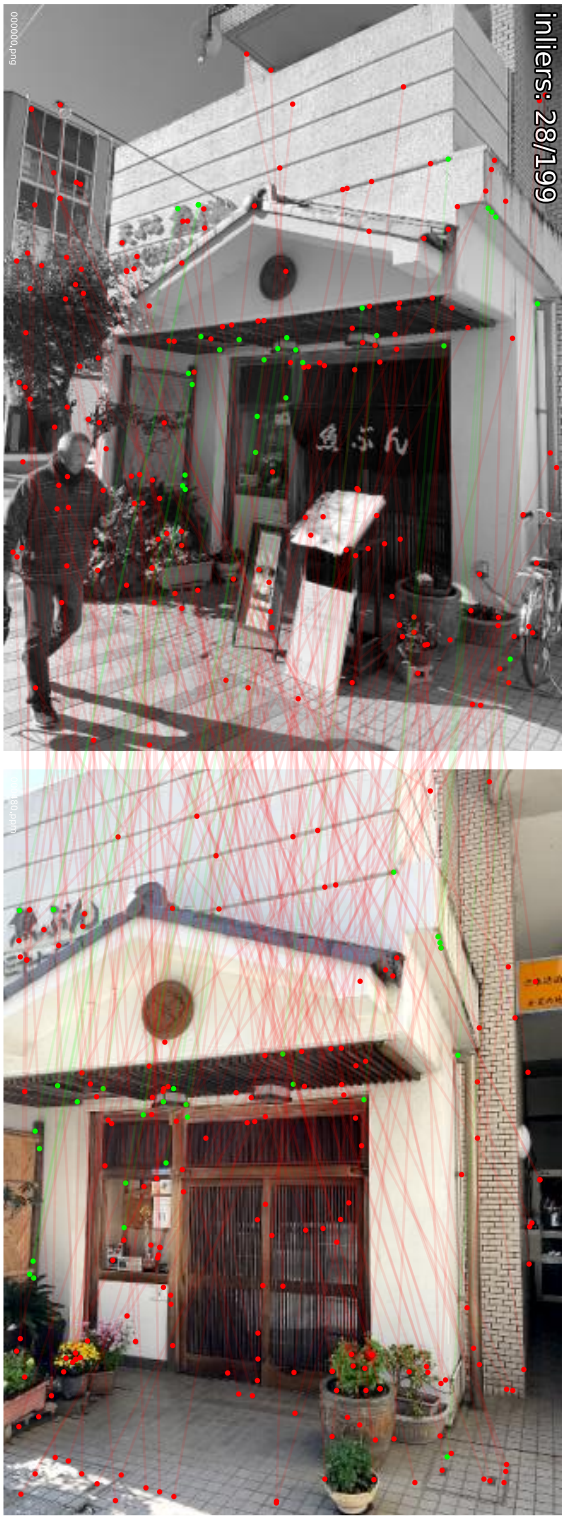


図 10-4 魚ぶん周辺での自己位置推定

4) デザートレストラン Grandma 上土本店周辺

- 車両向けシステム：測定不能
- スマートフォン向けシステム：誤差 35 ㎝
  - 実証中に扉の開閉があったが、問題なく自己位置推定ができた

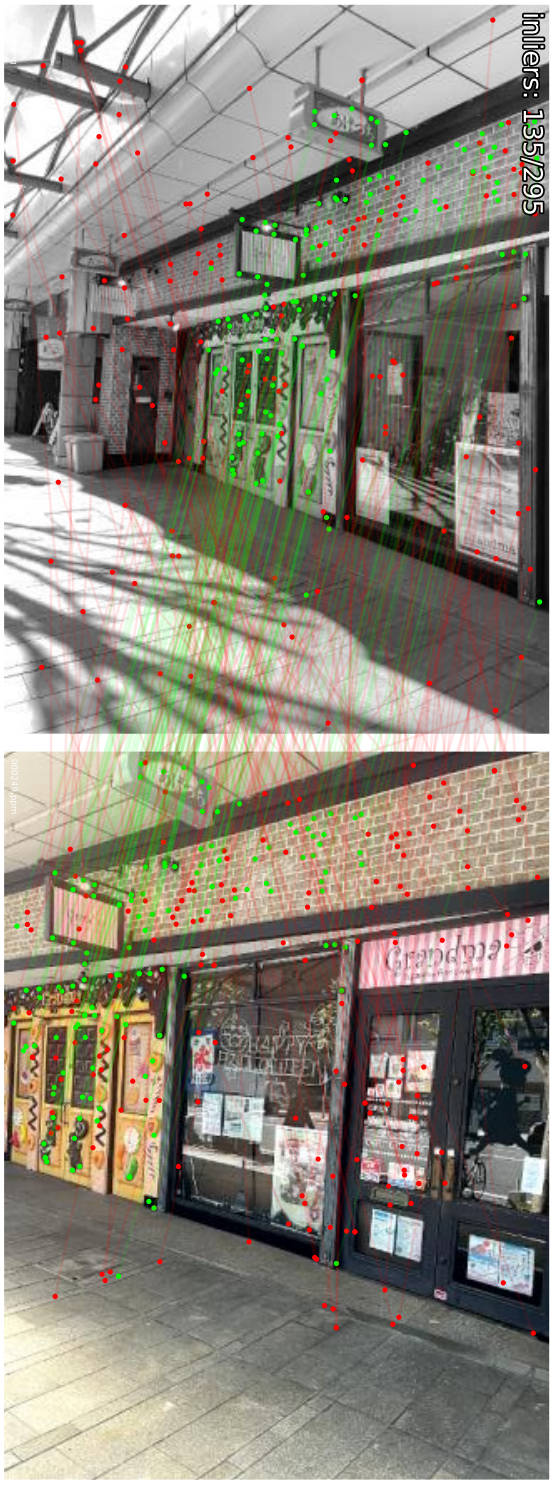


図 10-5 デザートレストラン Grandma 上土本店周辺での自己位置推定

5) 美容室　GUK 周辺

- 車両向けシステム：測定不能
- スマートフォン向けシステム：誤差 30 ㎝
  - 壁面に大きな特徴はなかったものの、看板や窓枠などをポイントに比較的高精度に自己位置推定できた

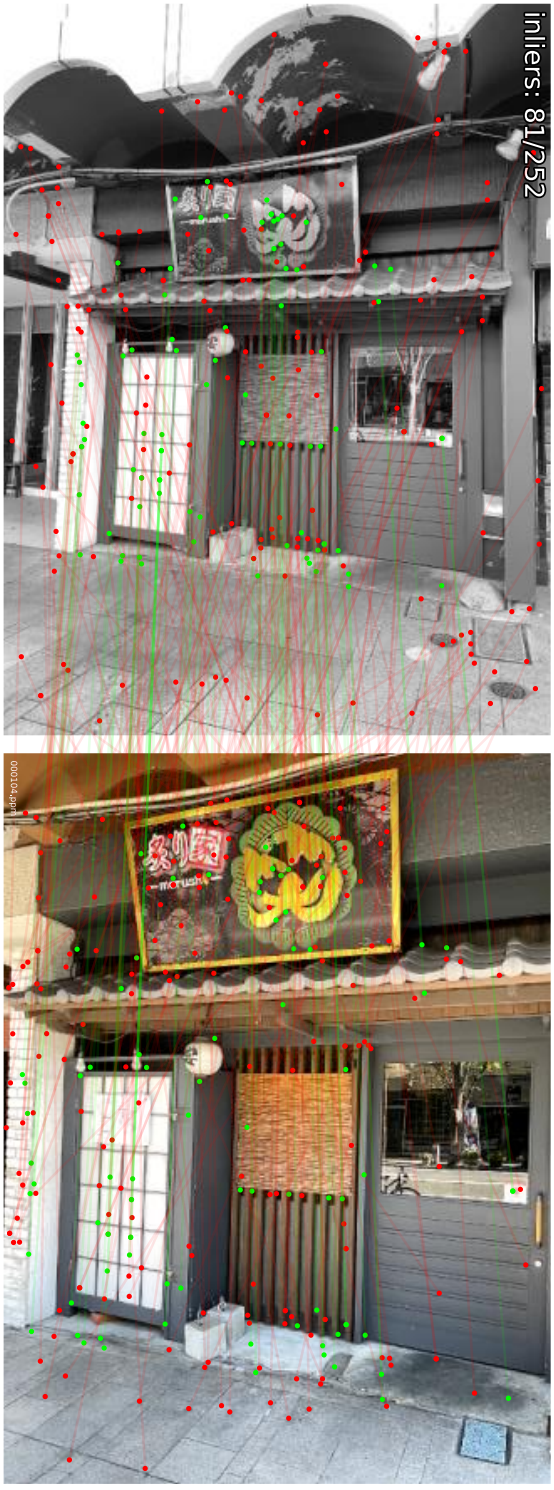


図 10-6 美容室　GUK 周辺での自己位置推定

6) サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺

- 車両向けシステム：誤差 300 ㎝
  - コントラストのはっきりした建物が有効であった

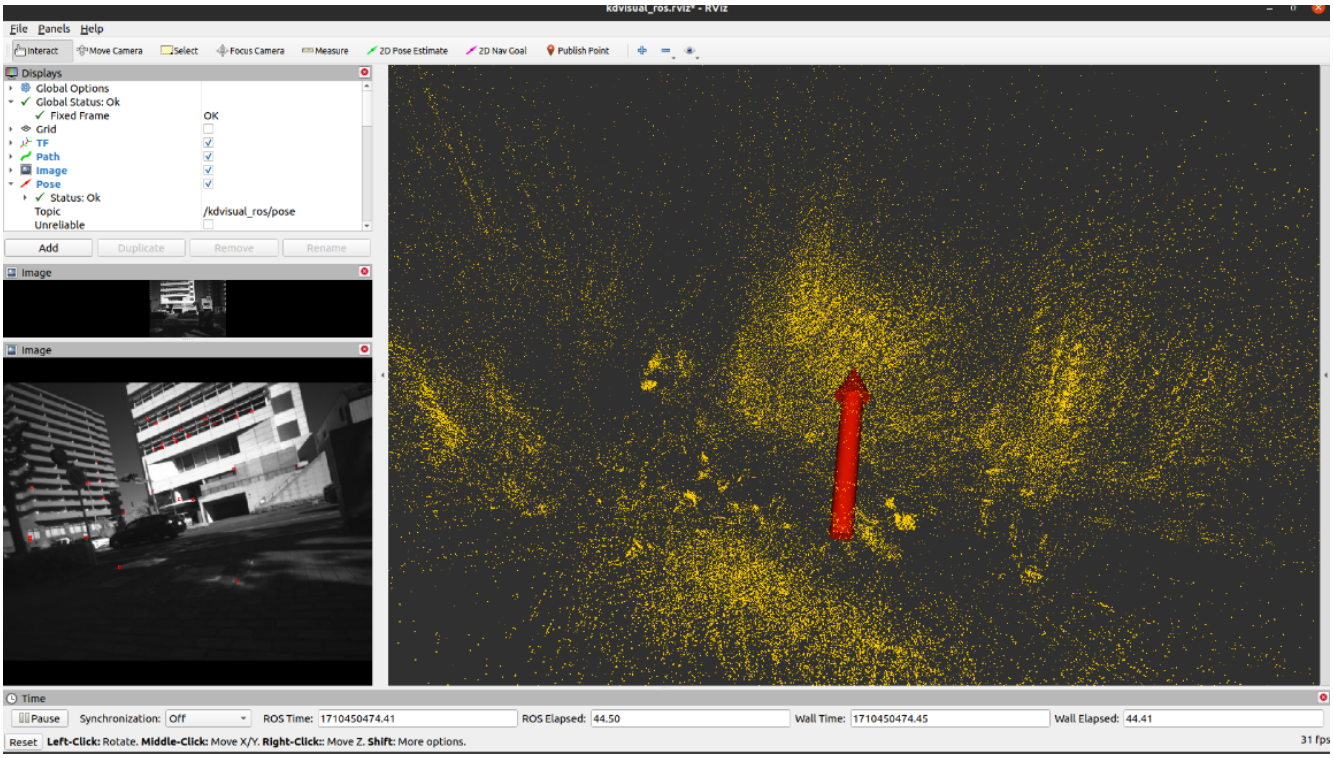


図 10-7 サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺での自己位置推定

- スマートフォン向けシステム：誤差 160 ㎝
  - 建築物が長方形のシンプルな外壁で覆われているため、高精度での自己位置推定には至らなかった

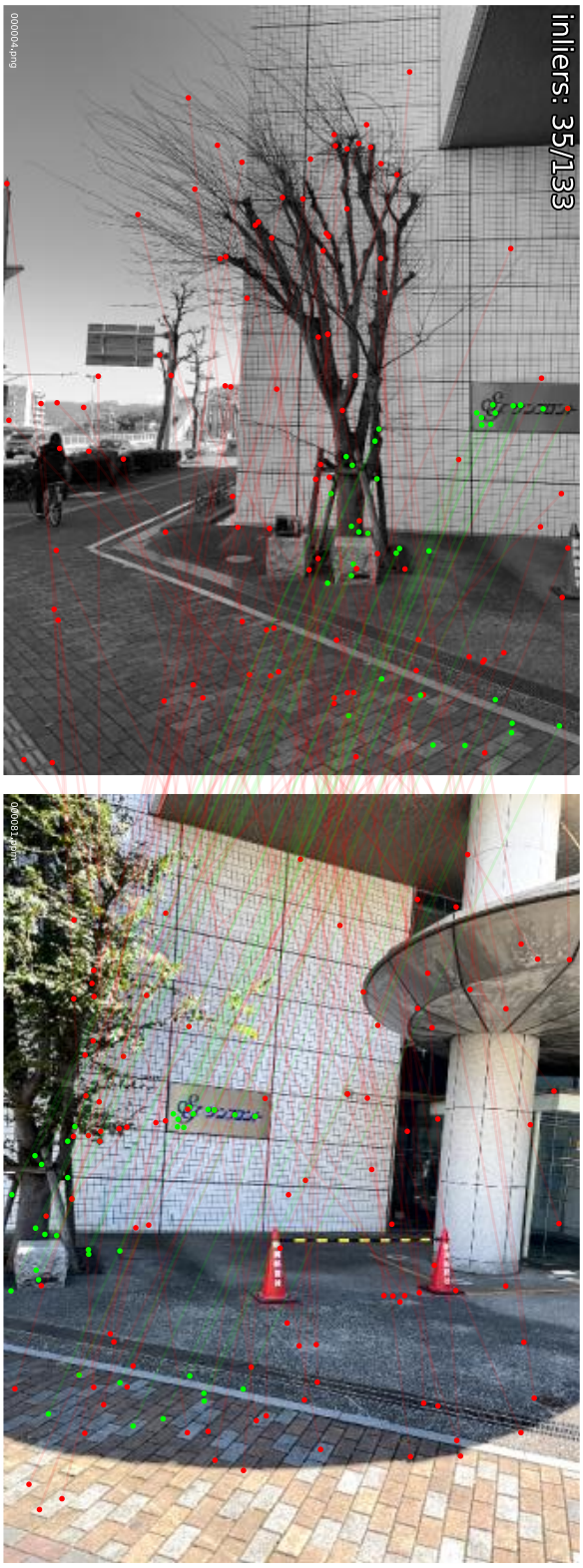


図 10-8 サンフロント静岡新聞ＳＢＳ静岡放送東部総局ビル周辺での自己位置推定

7) （有）一樹商事周辺

- 車両向けシステム：誤差 500 ㎝以上
  - 遠方のビルの特徴点もとらえているが、特徴点を絞り込むことができず、大きな位置ズレが発生した

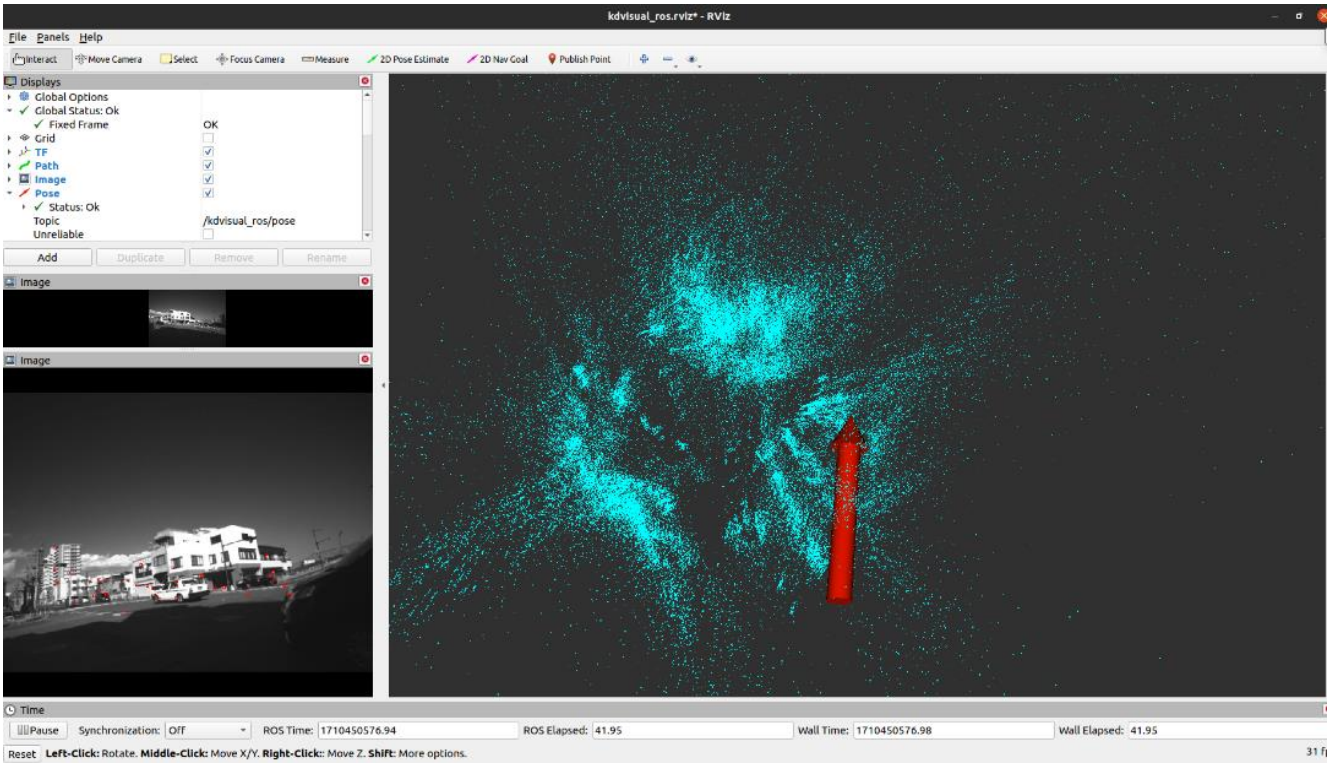


図 10-9（有）一樹商事周辺での自己位置推定

- スマートフォン向けシステム：誤差 40 ㎝
  - 窓枠や地面の模様などを優先的に特徴として得ることで自己位置推定を実現していた

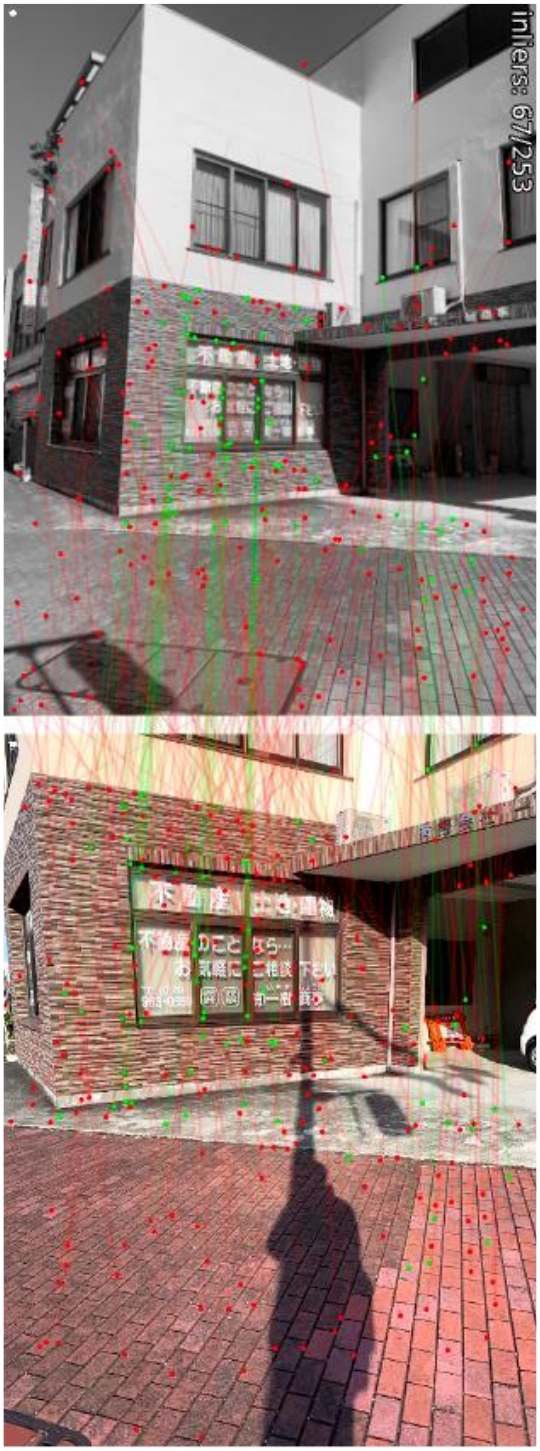


図 10-10 （有）一樹商事周辺での自己位置推定

8） 不動院周辺

- 車両向けシステム：誤差 500 ㎝以上
  - 建物の窓などの特徴点を捉えて自己位置推定を行っているが、特徴点を絞り込むことが出来ず似たような場所を特定してしまっている

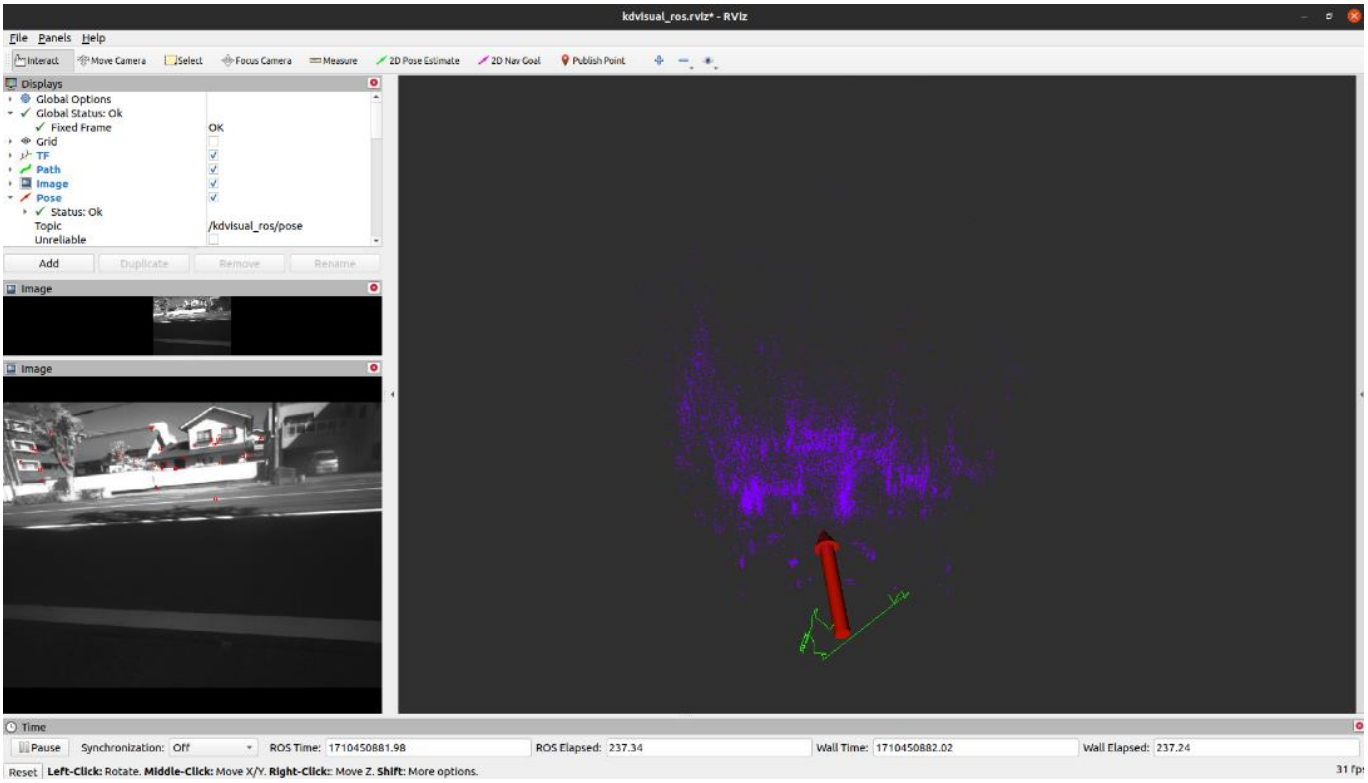


図 10-11 不動院周辺での自己位置推定

- スマートフォン向けシステム：誤差 40 ㎝
  - 道路から建物まで若干距離があるが、建物の梁などの特徴点を抽出し高精度な自己位置推定できた

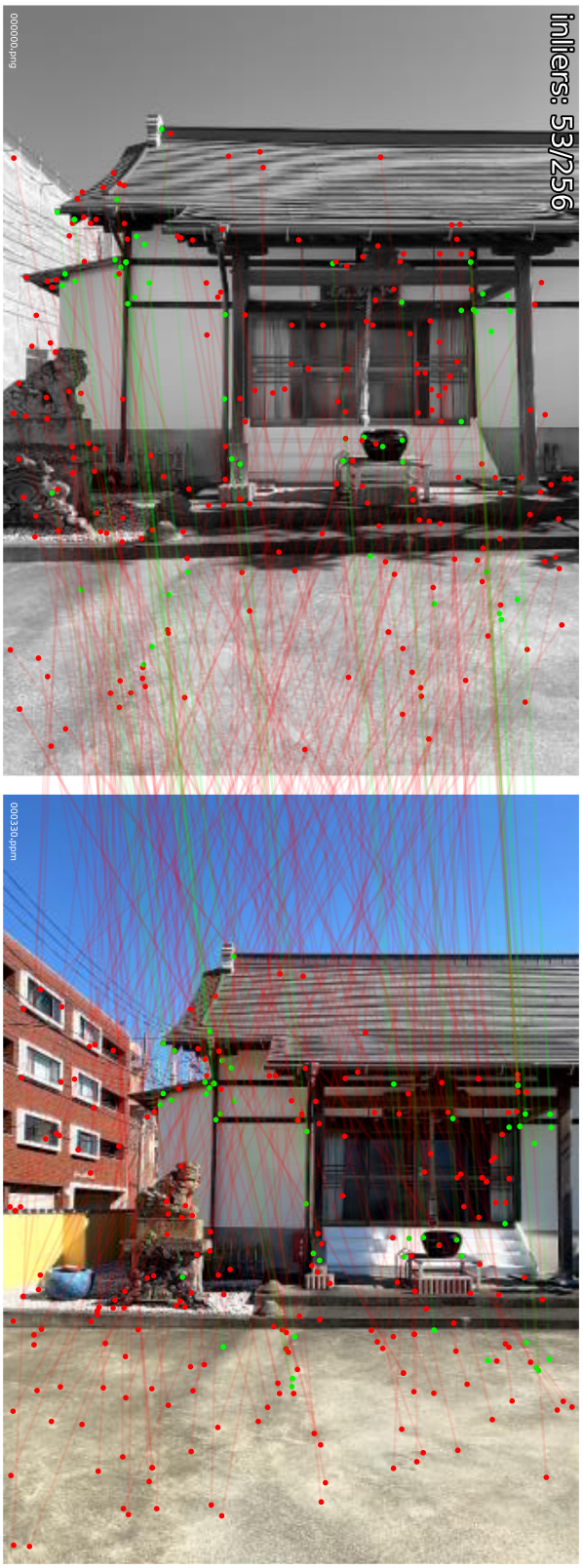


図 10-12 不動院周辺での自己位置推定

9) ファミリーマート 沼津下河原町店周辺

- 車両向けシステム：誤差 300 ㎝
  - 観測地点前後の特徴及び、遠方の特徴点も含めて自己位置推定できている

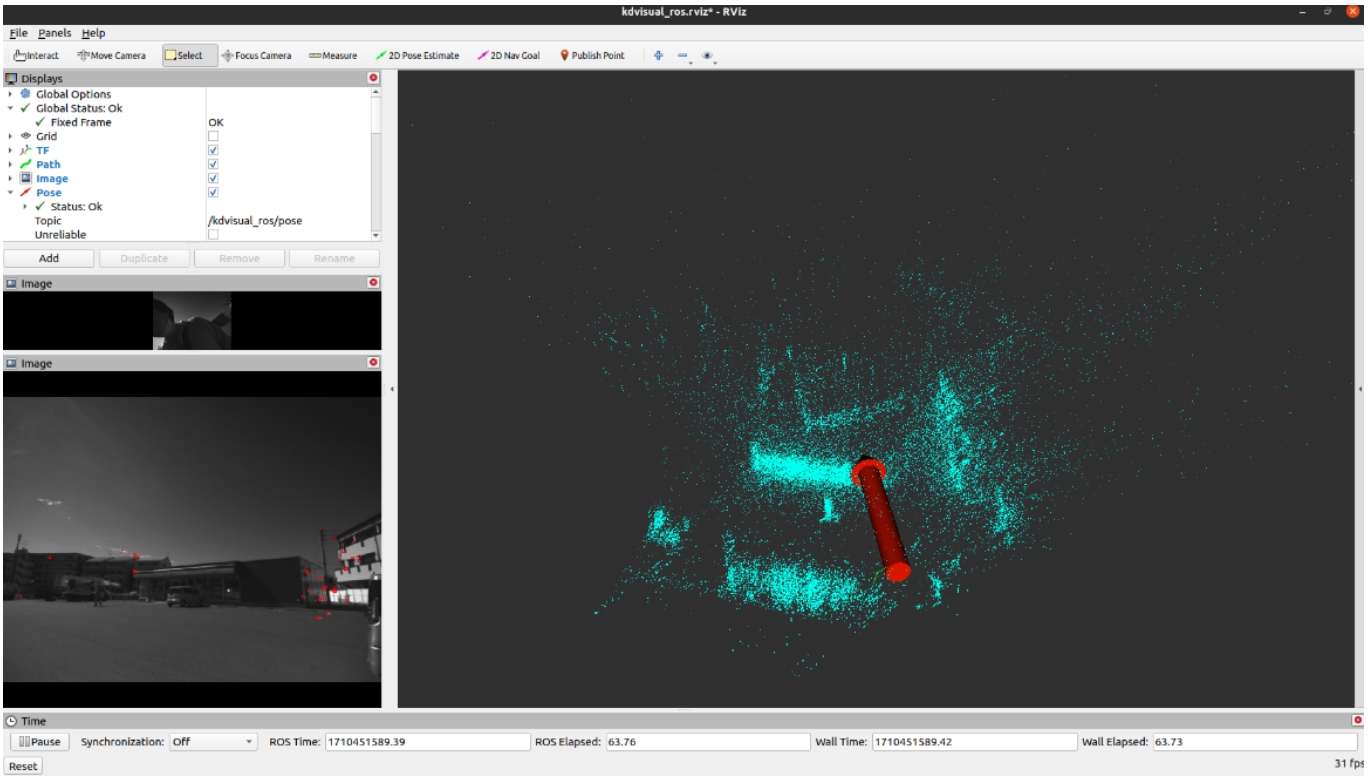


図 10-13 ファミリーマート周辺での自己位置推定

- スマートフォン向けシステム：測定不能
  - 自己位置推定処理は成功したものの、測定不能な精度誤差が発生した
  - 当初の想定どおりではあるものの、広い駐車場に囲まれており道路から建物までの距離がある、かつ周辺の建物と距離もあるため、共通の特徴が掴めず適切に自己位置推定できなかったと考える

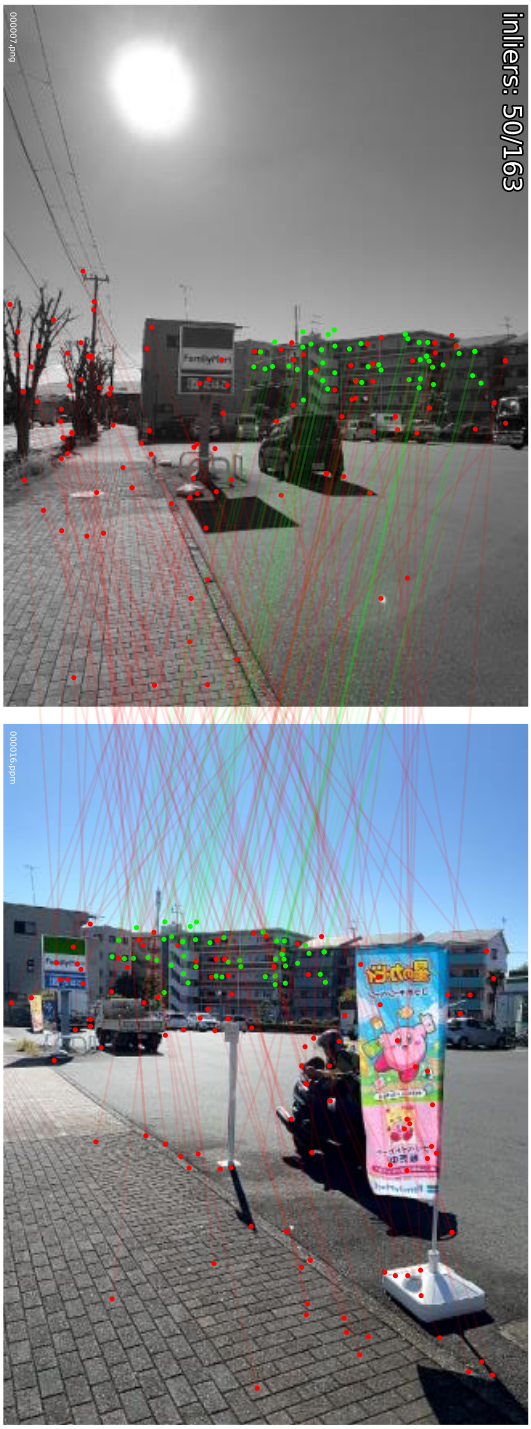


図 10-14 ファミリーマート周辺での自己位置推定

10）（有）するが水産周辺

- 車両向けシステム：誤差 100 ㎝
  - ➢ コントラストがはっきりした看板を特徴点として捉えて自己位置推定しており、移動に対する追従も比較的良好であった

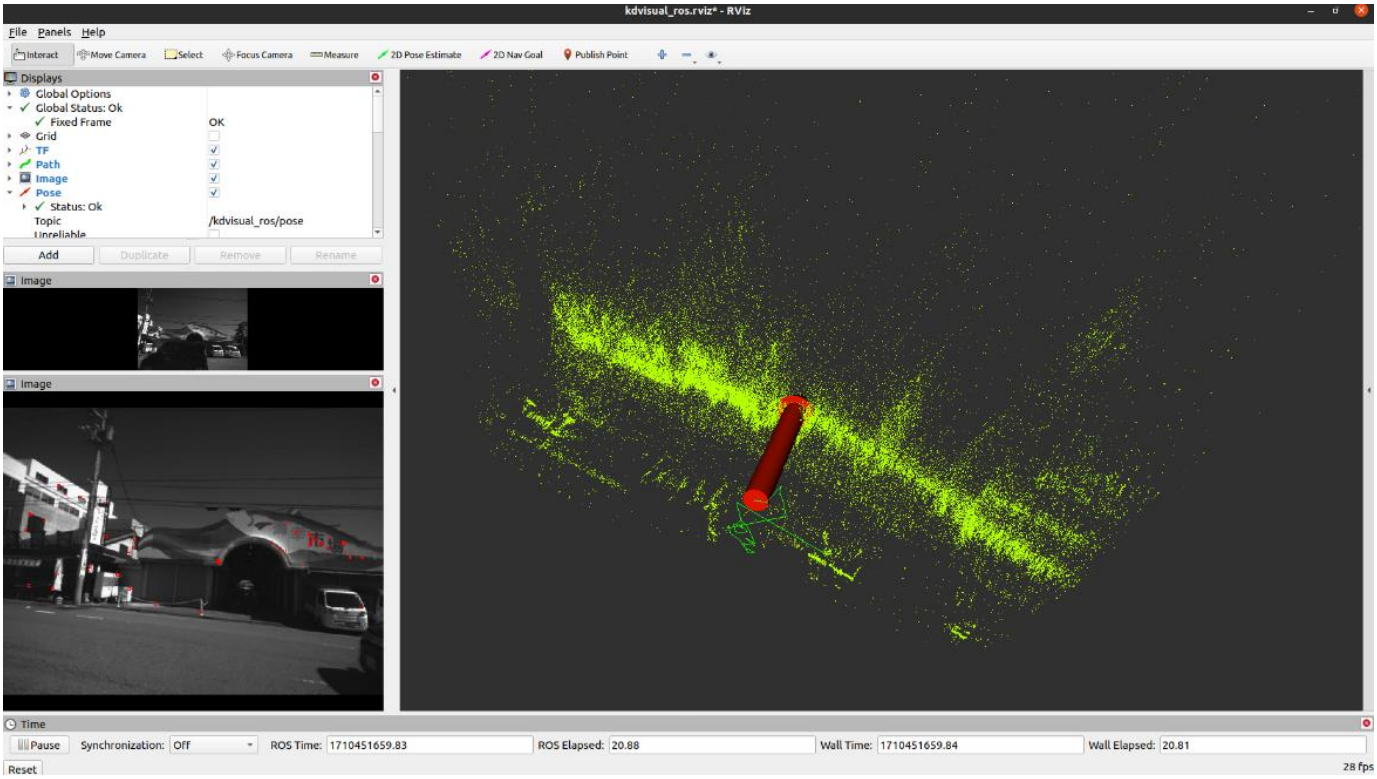


図 10-15 （有）するが水産周辺での自己位置推定

- スマートフォン向けシステム：誤差 70 ㎝
  - 人通りも多く環境変化が大きかったため、自己位置推定は期待よりは低い精度となったが、目標値はクリアできた

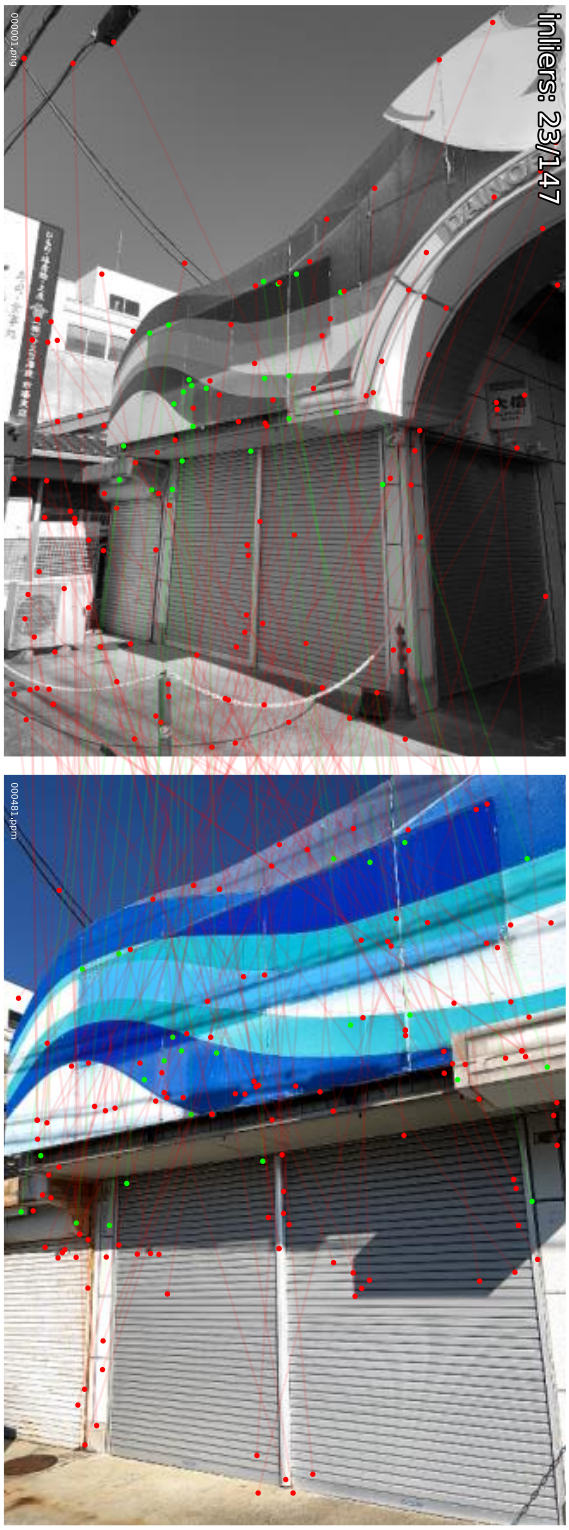


図 10-16 （有）するが水産周辺での自己位置推定

11. 両実証：成果と課題

11-1. 車両向け自律走行ステム：実証で得られた成果

11-1-1. 3D 都市モデルの技術面での優位性

実証実験を通じて、以下のような 3D 都市モデルの技術面での優位性が示された。

表 11-1 3D 都市モデルの技術面での優位性

大項目	小項目	3D 都市モデルの技術面での優位性
システム・機能	現在地の表示	● Status Display での LOD3 の 3D 都市モデル利用により、直感的に理解しやすい形で現在地を表示することができた
アルゴリズム	3D 都市モデルをレンダリングした画像からの VPS マップ作成	● 現地での撮影を必要としないことから何度でも条件を変えてマップ作成を試行できる ● マップ作成の自動化ができる

11-1-2. 3D 都市モデルのビジネス面での優位性

実証実験を通じて、以下のような 3D 都市モデルのビジネス面での優位性が示された。

表 11-2 3D 都市モデルのビジネス面での優位性

大項目	小項目	3D 都市モデルのビジネス面での優位性
サービスの提供価値向上	簡易に VPS 技術を使用可能	● 3D 都市モデルを VPS マップに活用できることで、気軽に VPS を使ったコンテンツ開発が可能となる
	各種サービスの基盤として有用	● 小規模モビリティやスマートフォン用 VPS として、カメラ画像のみで自己位置推定可能なシステムは有用 ● ナビゲーションや AR コンテンツのための基盤となる ● デジタルツインのために、デジタルと現実をつなぎ合わせる技術として有用
サービス開発期間・コストの削減	開発工数・コストの削減	● 現地での撮影が必須となる VPS マップの作成時の工数・コストが削減可能
	オープンデータによる開発・運用コスト削減	● 3D 都市モデルは公的なオープンデータとして整備されていることから、整備範囲での VPS マップを追加コストや許可手続きを経ることなく入手可能
	整備範囲の広さによるビジネスの拡張性	● ビジネスとしてのスケールを考えた場合、3D 都市モデルの全国的な整備が進むことで特定の地域に限らない活用が可能

11-1-3. 3D 都市モデルの政策面での優位性

実証実験を通じて、以下のような 3D 都市モデルの政策面での優位性が示された。

表 11-3 3D 都市モデルの政策面での優位性

大項目	小項目	3D 都市モデルの政策面での優位性
その他	都市のデジタル化基盤	● 都市内の場所を端末から認識可能にする VPS は、都市内の位置にひも付く情報を管理する基盤となる ● 3D 都市モデルから VPS マップを作成することで、容易に都市内の位置にひも付けた情報を管理することができ、新たな都市サービスにつながる

11-2. 車両向け自律走行システム：事業化に向けた課題と解決策の案

11-2-1. 事業化に向けた課題

表 11-4 事業化に向けた課題と解決策の案

大項目	小項目	実証実験で得られた課題	課題に対する対応策
システム（機能）	システム構成の改善	● 複数のコンポーネントがネットワークなどを介して複雑に接続しているため、運用難度が高い	● 接続方法の変更や必要なプログラムの再実装などで 1 機器 1 プログラムに完結して動作するようなシステムに修正する
アルゴリズム	特徴点のマッチング精度の向上	● KdVisual では、3D 都市モデルをレンダリングした画像と実写画像との特徴点の照合に課題があった ● 両者がマッチングせず自己位置推定ができなかった	● 新規のアルゴリズムの開拓など、抜本的な改善が必要（今後の展望に詳細を記載）
	動作速度の向上	● データ通信のボトルネックなどから、動作速度が十分でなかったことも精度に悪影響を与えている可能性がある	● システム構成を見直し、より高速な動作を可能にする

11-2-2. 解決策の案・今後の展望

まず、3D 都市モデルの詳細度を向上させる方向性があるが、データ作成コストの上昇や更新頻度の課題、データサイズの巨大化などの理由により、現実的な選択肢ではない。このため、レンダリング画像と実写画像をより高度に照合するアルゴリズムが必要と考える。

今回の実証では、C*・KdVisual どちらのシステムも比較的古典的な照合アルゴリズムを利用したものの、今後においては機械学習アルゴリズムの利用は検討に値する。

本検証で使用しているアルゴリズムは 2 種類の系統に分かれており、C*が採用している両画像のピクセルごとの比較を全て行い画像全体での類似度を評価する手法と、Immersal（2021 年度使用）と KdVisual（今年度）が採用している、画像の特徴的な点を抽出しその特徴ベクトルを比較する手法である。

近年、特徴点を照合する手法では、機械学習を用いたより高度な手法が提案されている。今後の展望の検証のため、実際に照合した結果を以下に示す。

【比較元：照合アルゴリズム（SIFT 特徴量 総当たりマッチング)】

使用した画像は、今回の検証の過程で RealSense D455 カメラにより取得した画像と Unity 上で 3D 都市モデルをレンダリングした画像である。まず、古典的な手法である SIFT 特徴量[12]総当たりマッチングの結果を図 11-1 に示す。左右の画像を結ぶ線が、アルゴリズムにより同一点と判断された対の点であるが、多くの点で誤った点同士を組み合わせてしまっている。

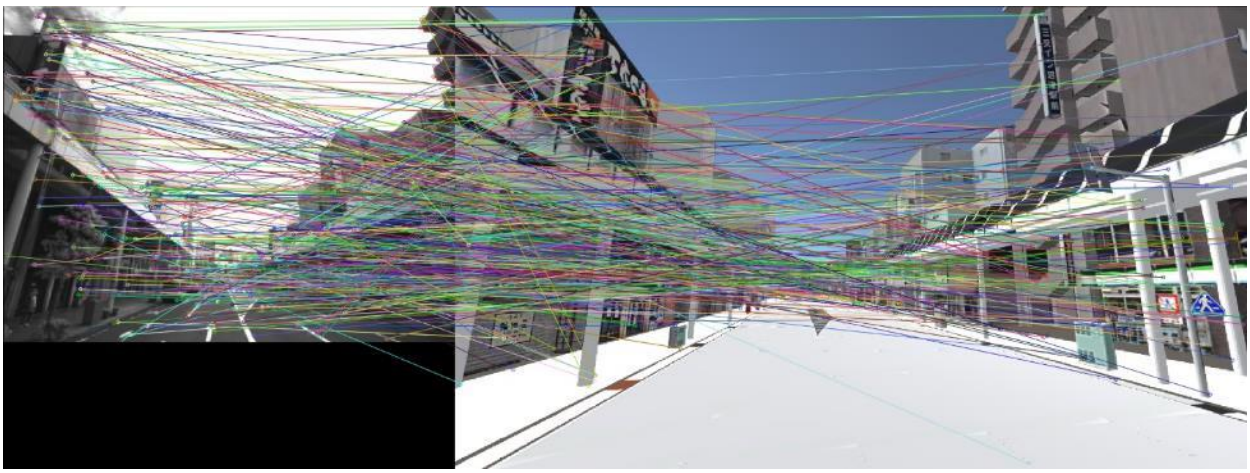


図 11-1 SIFT での特徴量マッチング

【比較先：機械学習アルゴリズム（SuperPoint と LightGlue を用いた特徴点検出）×照合アルゴリズム】

一方で図 11-2 は、SuperPoint[13]と LightGlue[14]という機械学習を用いた特徴点検出と照合のアルゴリズムで同様の処理を行った結果であり、前者と比較すると一定正確な正確な照合結果となっている。

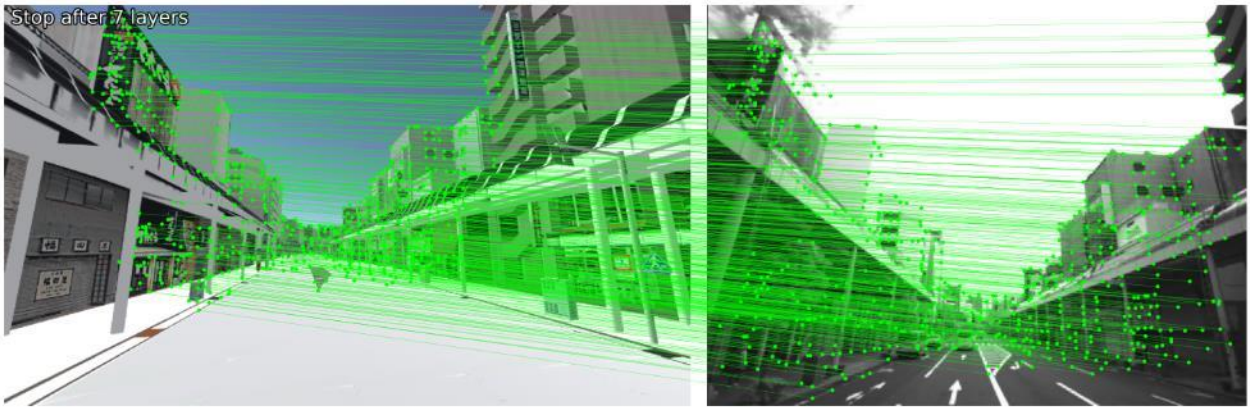


図 11-2 SuperPoint+LightGlue での特徴量マッチング

特に LightGlue では、Positional Encoding を利用して点の位置関係も学習データとして投入しているため、特徴量の類似度だけで照合するよりも良好な結果が出ているのではないかと考えられる。

[12] Lowe, David G. (1999). "Object recognition from local scale-invariant features"

[13] https://github.com/rpautrat/SuperPoint

[14] https://github.com/cvg/LightGlue

【SuperPoint+LightGlue で照合した結果を利用したカメラ位置計算】

さらに、レンダリングした画像の特徴点の座標を逆に Unity 空間に投影することで、各特徴点の三次元座標を取得できる。ここから基本的なカメラ位置の推定アルゴリズム（VPS の基本的なアルゴリズム）を適用したところ、図 11-3 のように実際の位置に近い座標が得られた。

左下推定座標からのレンダリング画像、右検証用実写画像がほぼ同様の画角となっていることがわかる。

しかし、複数のデータでの検証を行った結果、大きく位置推定結果がずれる地点も多くあり、この手法だけで解決する課題ではないことがわかった。

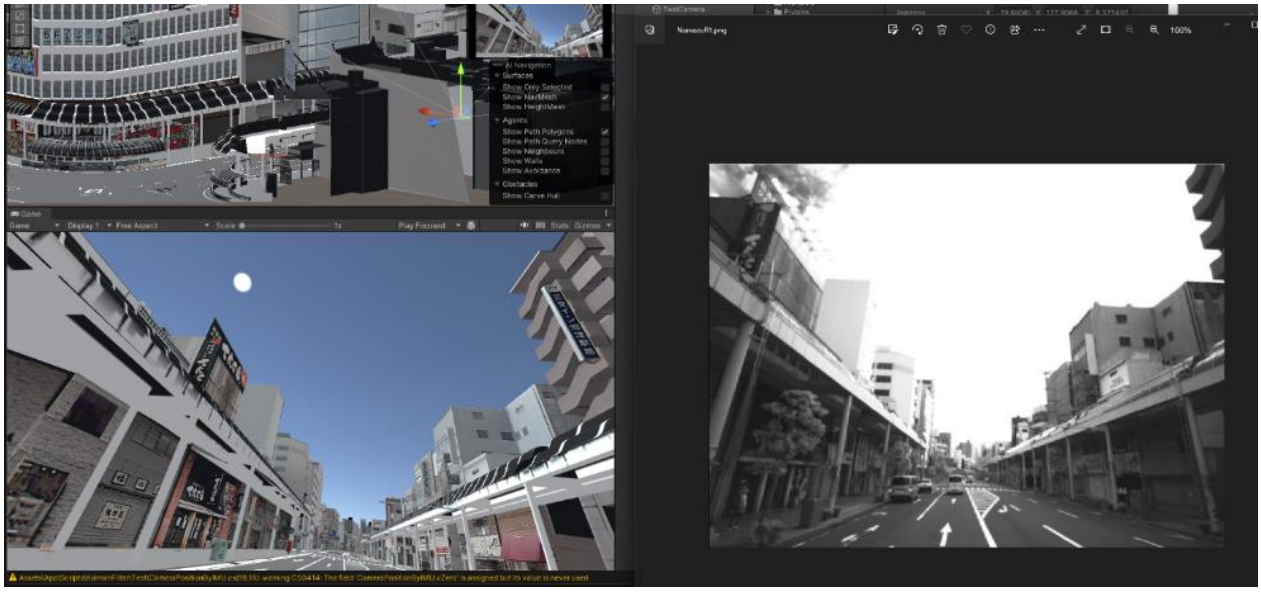


図 11-3 VPS 計算を行い Unity 内のカメラ位置に反映した結果画像と実写の比較

また、機械学習を活用したアルゴリズムは学習データにより性質や性能が大きく変わる。現在の画像照合用の機械学習アルゴリズムは、実写画像同士の照合に合わせて学習しているが、これを 3D 都市モデルのレンダリング画像と実写画像を学習データとすることでさらなる性能の向上が期待できる。

【今後の調査の方向性と展望】

さらに、本実証で検証してきた「KdVisual」および「Immersal」のような特徴点で照合するタイプと、「C*」のように画像全体での類似度を見るタイプの二系統のアルゴリズム以外にも画像からのカメラ位置推定アルゴリズムは研究されており、画像内の線分を検出して推定を行う LIMAP[15]や、NeRF[16]や Gaussian Splatting[17]などの空間表現で活用されている微分可能レンダリングを使ったカメラ位置推定手法などの研究もある。こうした新規のアルゴリズムは、今回のアルゴリズムとは認識・判断している対象の特徴が異なるため、3D 都市モデルと実写との対照による自己位置推定を可能とするものがあるかを検討することも必要であり、より高い性能で照合が可能なアルゴリズムの開発と試行が必要であると考える。

[15] https://github.com/cvg/limap

[16] arXiv:2003.08934

[17] https://github.com/graphdeco-inria/gaussian-splatting

11-3. スマートフォン向けアプリケーション：実証で得られた成果

11-3-1. 3D 都市モデルの技術面での優位性

実証実験を通じて、3D 都市モデルを適用した VPS 技術について、以下のような技術面での優位性が示された。

表 11-5 3D 都市モデルの技術面での優位性

大項目	小項目	3D 都市モデルの技術面での優位性
自己位置推定処理	精度	● 3D 都市モデルをもとに、おおよそ 1m 以下の位置誤差で自己位置推定を実施することができた
	性能	● 3D 都市モデルをもとに、おおよそ 10 秒以下の時間で自己位置推定を実施、AR コンテンツを表示することができた
AR アプリケーションの開発	容易さ	● 3D 都市モデルをもとに AR アプリケーションを開発することで、現地でのスキャン作業なく AR アプリケーションを開発することができた

11-3-2. 3D 都市モデルのビジネス面での優位性

実証実験を通じて、3D 都市モデルを適用した VPS 技術について、以下のようなビジネス面での優位性が示された。

表 11-6 3D 都市モデルのビジネス面での優位性

大項目	小項目	3D 都市モデルのビジネス面での優位性
サービスの提供価値向上	気軽に VPS を活用可能	● 3D 都市モデルを VPS マップに活用できることで、気軽に VPS を使ったコンテンツ開発が可能となる
	各種サービスの基盤として有用	● 小規模モビリティやスマートフォン用 VPS として、カメラ画像のみで自己位置推定可能なシステムは有用 ● ナビゲーションや AR コンテンツのための基盤となる ● デジタルツインのために、デジタルと現実をつなぎ合わせる技術として有用
サービス開発期間・コストの削減	開発工数・コストの削減	● 現地での撮影が必須となる VPS マップの作成時の工数・コストが削減可能
	オープンデータによる開発・運用コスト削減	● 3D 都市モデルは公的なオープンデータとして整備されていることから、整備範囲での VPS マップを追加コストや許可手続きを経ることなく入手可能

	整備範囲の広さによるビジネスの拡張性	● ビジネスとしてのスケールを考えた場合、3D 都市モデルの全国的な整備が進むことで特定の地域に限らない活用が可能

11-4. スマートフォン向けアプリケーション：事業化に向けた課題と解決策の案

11-4-1. 事業化に向けた課題

今回の実証実験において、点群 to 点群対応化アルゴリズムの完全な自動化は実現できていない。これは、前述のとおり 3D 都市モデルから生成した点群と現実の世界から生成した点群とでデータの範囲やクオリティに大きなギャップが存在することに依存していると考えられる。
最終的にユーザーアプリとして広く展開するためには、ユーザーによる自己位置推定に必要なオペレーションの負荷低減や、自己位置推定やトラッキング機能により継続的にユーザーの位置情報を提供する仕組みを提供する必要がある。

表 11-7 実証実験で得られた課題

大項目	小項目	実証実験で得られた課題	課題に対する解決策の案
システム	精度	● 点群 to 点群対応化アルゴリズムの完全な自動化に至っていない ● 環境によって精度は大きく低下するケースが存在する	● 3D 都市モデルの持つ面や線など、点群以外のデータを活用する。 ● 3D 都市モデルの各頂点やポリゴンの法線（Normal）ベクトルを活用する ● ユーザーのオペレーションによる対応化処理のサポート機能を設計する
	性能	● 環境やユーザーのオペレーションによって処理時間が長くなるケースが存在する	● アプリケーション上でユーザーのオペレーションをサポートするコンテンツを作成する
利便性	オペレーション	● ユーザーが現場で何度もスキャンを実施する手間が発生する	● トラッキング機能を活用することで、1 度のスキャンで対応できるエリアの範囲を拡大し、スキャン頻度を低減する ● アプリケーション上でユーザーのオペレーションをサポートするコンテンツを作成する ● スキャンオペレーションにゲーム要素などユーザーへの価値を負荷することでスキャン処理の精神的な負荷を低減する
	3D 都市モデルの整備	● LOD3 以上の都市モデル整備エリアが少なく、利用できるエリアが限定的	● 3D 都市モデルの LOD3 整備エリアを拡大する ● 3D 都市モデルのテクスチャ作成方

		● 3D 都市モデルの品質にばらつきが存在し、精度に大きく影響を与える	式を確立・統一する

11-4-2. 解決策の案

本項では、システム及びユーザーの利便性という 2 つの観点に分けて課題の解決策の案を示唆する。

観点①システム

今回の実証において、精度・性能ともに場所への依存はあるものの、おおよそ設定した KPI を達成することができた。一方で、点群 to 点群対応化アルゴリズムにおいて、完全な自動化には至らず手動による対応化処理を含んでいる。

今後、本システムを実際に利用するためには、システム管理者による手動処理を排除する必要がある。また、環境やユーザーのオペレーションによって処理時間が長くなるケースが存在することが判明した。アンケート結果からも処理時間は非常に重要な要素であり、処理時間にばらつきがある不安定なシステムはユーザーがサービスを利用する際の大きなハードルとなる。

- 点群 to 点群対応化アルゴリズムの自動化

【課題】

今回の実証実験において、3D 都市モデルから出力した点群と現実世界から出力した点群の対応化処理において、自動かつ高精度での対応かを実現することができなかった。これは、過去時点の情報をもとに作成し、かつ現実と異なる建物や道路地物などのテクスチャが貼られることのある 3D 都市モデルと、現実世界に大きな乖離が存在するためと考える。

そのため、今回の実証実験では最終的に下記の点群 to 点群対応化アルゴリズムを適用し手動による補正を取り入れた。ここで、図 11-4 で記載した Step1 において手動により 4 点の特徴点のマッチングを実施している。(図 11-5 参照)。これはもとのデータである現実世界から取得した点群が示すエリアの範囲（ユーザー視点）と 3D 都市モデルから取得した点群が示すエリアの範囲（エリア全体）による差や、3D 都市モデルのテクスチャの品質から点群情報だけをもとに対応化することの難易度が高く、手動により対応化処理をサポートしたものである。

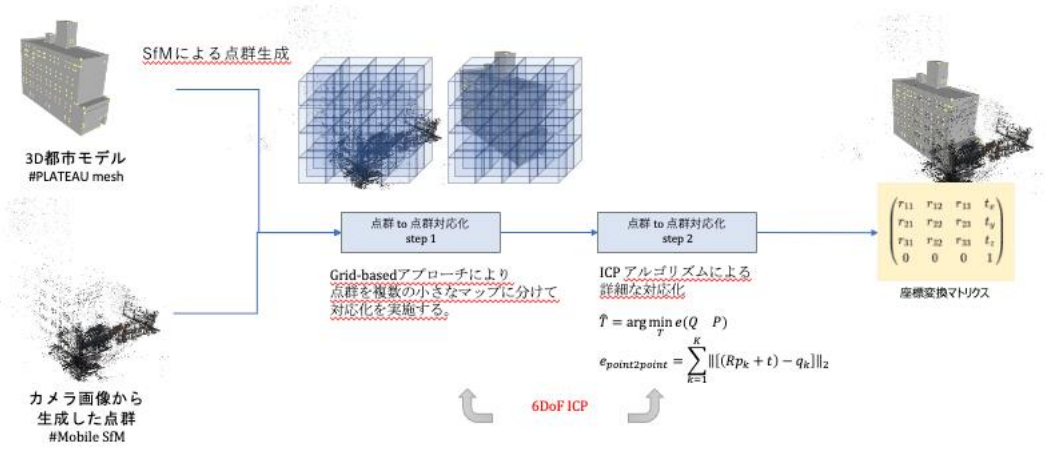


図 11-4 アルゴリズム 2 における点群 to 点群対応化（再掲）

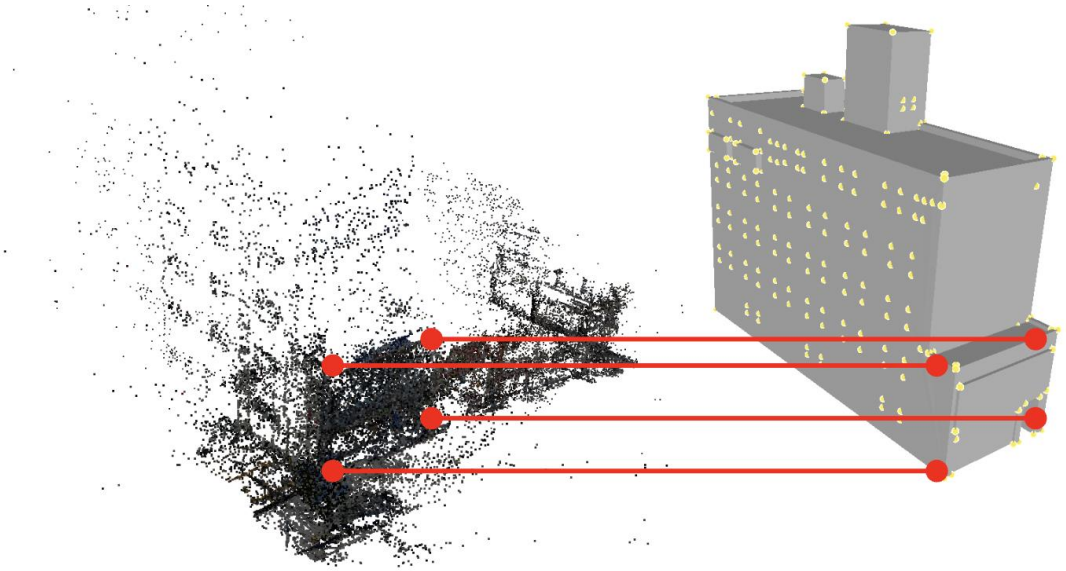


図 11-5 アルゴリズム 2 における点群 to 点群対応化の手動対応

【解決策】

3D 都市モデルには線や面、法線など様々な情報を保持している。今後、これらより多くの情報を組み合わせることで、テクスチャなどの視覚的な情報として不足する情報を補い、手動による処理なく、対応化処理の実現が期待できる。また、さらにユーザーがアプリケーションを利用する際に、アプリケーション上で透過された 3D 都市モデルを表示しオペレーションをサポートすることにより、より高精度での対応化の実現が期待できる（図 11-7 参照）。

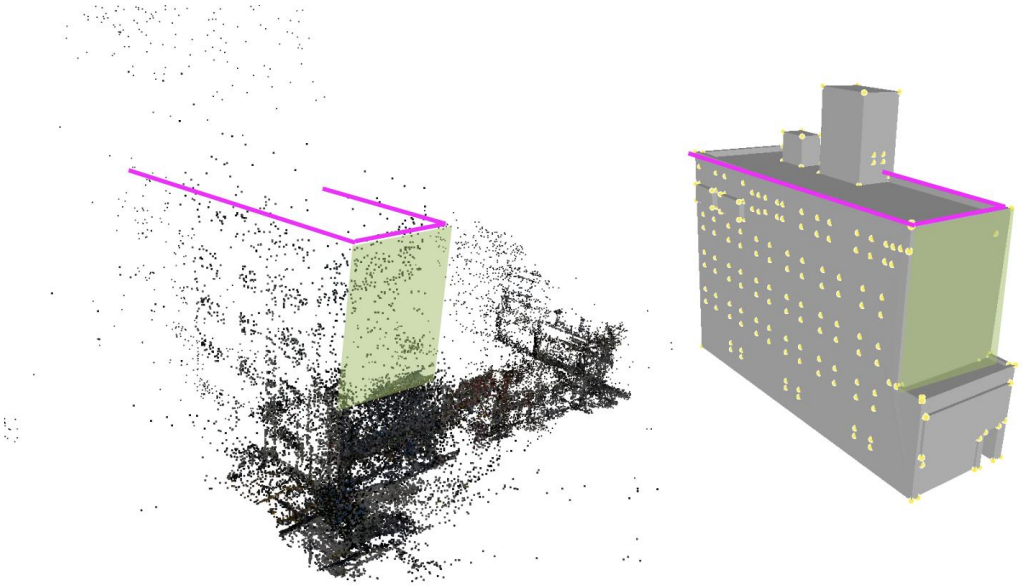


図 11-6 3D 都市モデルの面や線、各頂点やポリゴンの法線（Normal）ベクトルの活用イメージ

図 11-7 ユーザーオペレーションにより 3D 都市モデルと現実世界を対応化させるイメージ

- 環境による精度のばらつき

【課題】

今回の実証により、現場の環境により VPS の精度が大きく低下するケースが存在した。本事象は先に示したとおり、3D 都市モデルの利用有無に関わらず、従来からの VPS に対する課題である。本事象についてアルゴリズムにより急速かつ大幅に改善することは期待できない。

【解決策】
前述（図 11-7 参照）のようなユーザーがアプリケーションを利用する際に、アプリケーション上で透過された 3D 都市モデルを表示しオペレーションをサポートする機能を実装することで、ユーザー自身が VPS の得意な領域へカメラを向けることを促し、VPS が苦手とする環境を避けながらアプリケーションを利用することで、環境による精度のばらつきを抑え安定したシステムを提供することが期待できる。

- 環境やユーザーオペレーションによる処理時間のばらつき

【課題】
精度同様に、処理時間においても現場の環境や、カメラの向きや動かし方といったユーザーオペレーションの影響を受け、処理時間にばらつきが発生した。

【解決策】
本課題においても前述のようなユーザーがアプリケーションを利用する際に、アプリケーション上で透過された 3D 都市モデルを表示しオペレーションをサポートする機能を実装することで、VPS が苦手とする環境を避けながらアプリケーションを利用し、環境よる精度のばらつきを抑え安定したシステムを提供することが期待できる。（図 11-7 参照）

観点②ユーザーの利便性

【課題】
今回の実証実験のアンケート結果から、ユーザーはアプリケーション上でコンテンツを表示するまでの「時間」を非常に重視していることがわかった。本システムでは、実際にユーザーがアプリケーションを利用するにあたり、専用アプリを利用した複数回のスキャン作業が必要となる。つまり実際にサービス化していくためには、このスキャン作業によるユーザーの負荷を低減することが非常に重要な課題となる。一方で、現状のアプローチにおいて現場のスキャンは必須なものとなる。

【解決策】
今後、UI/UX の観点からオペレーションのアシストや、スキャンデータのスコアリングに応じた景品やポイントの付与、ゲーム要素を取り入れるなどの対応により、ユーザーへ新たな価値を提供することで煩雑なスキャンをスムーズかつポジティブな体験に変えていくことが期待できる。

このように、本システムを社会へ展開するためには、システムそのものの精度や性能にとどまらず、適切な用途やユーザー体験を合わせて検討していくことが重要であると言える。

11-5. 今後の展望

今回の実証実験では、3D 都市モデルを活用した VPS 技術の確立と社会実装に向けた汎用性の拡大という 2 つの観点で、昨年度開発した「C*」に「KdVisual」を組み合わせた車両向け自律走行システムの高度化と「PretiaVPS」をベースとしたスマートフォン向けのアプリケーションとして利用可能な VPS 技術を構築した。

車両向け自律走行向けシステムについては、事前に現地に訪問の上、多数の画像を撮影してマップを作成する必要があるという VPS の課題に対して、ゲームエンジン上でレンダリングした 3D 都市モデルの画像から VPS マップを作成することは成功したが、そのマップを活用して現実世界のカメラ画像とのマッチングによる精度の高いローカライズは困難であった。

3D 都市モデルを活用した VPS 技術の確立のためには、既存の VPS アルゴリズムを流用するのではなく、新規に 3 D 都市モデルに適した VPS アルゴリズムを開発することが必要と考える。例えば、3D 都市モデルのレンダリング画像と現実世界のカメラ画像を教師データとして準備し、SuperPoint や LightGlue のような最新の特徴点抽出マッチングアルゴリズムを機械学習させ、特徴点の照合に必要なパラメータの選定や閾値を算出することで 3D 都市モデルに特化した VPS アルゴリズムの確立が可能になると想定される。新たな VPS アルゴリズムにより、高精度かつ安定した自己位置推定が実現されれば、3D 都市モデルが整備されている場所全域で使える VPS 技術となり、モビリティや XR、デジタルツインの基盤など、デジタルとリアルを融合する基盤技術として広範囲の応用が見込める。

一方で、スマートフォン向けのアプリケーションとして利用可能な VPS 技術の実証実験では、PLATEAU SDK for Unity をもとに Point Cloud Registration 技術を応用した座標変換マトリクスを出力する「点群 to 点群対応化機能」を開発した。一部手動操作が必要ではあるが、自動化を見据えた対応付けの仕組みを構築でき、定量的・定性的にも十分な性能の VPS 機能が実現できた。

更なる実用化に向けては、精度や性能の改善はもとより、「点群 to 点群対応化機能」の完全な自動化や、実際にナビゲーションとして利用を想定し、ユーザーエクスペリエンスの向上に向けて、LOD4 や BIM データと連携し、屋内・屋外などの利用可能範囲の拡大が必要になると考えられる。これらの機能改善により、将来的には屋内外をシームレスに案内することが可能な一般ユーザー向け観光ナビゲーションや、ビジネスシーンにおいてはラストワンマイルモビリティの高度化を期待することが可能となり、移動や配送における 3D 都市モデルの利用用途拡大を目指す。

12. 用語集

A)　アルファベット順

表 12-1 用語集（アルファベット順）

No.	用語	説明
1	ADENU	Autonomous Drive Enabler by Nagoya University の略で、名古屋大学で 開発された自動運転車用のソフトウェアパッケージ。自動運転などの自動走行に必要 となる機能（各種センサデータの取得、地図データ管理、地図とセンサを融合した 走行状況の把握等）が体系的に含まれたソフトウェア
2	AR	Augmented Reality の略称。現実世界に仮想世界を重ね合わせて表示する技術
3	BFGS 法	Broyden–Fletcher–Goldfarb–Shanno algorithm の略。準ニュートン法の一種。非線形最適化問題の反復的解法の一つ。
4	C*	産業技術総合研究所で開発された SLAM/VPS 技術。
5	CSV	Comma Separated Value の略。コンマで区切られたテキスト形式のデータフォーマット。
6	DEM	Digital Elevation Model、数値標高モデル、地表面の地形のデジタル表現
7	Feature Extraction	画像上から特徴点を抽出するデータ処理技術
8	Feature Matching	特徴点のマッチングを行うデータ処理技術
9	Geometric Verification	マッチングした特徴点の妥当性を幾何学的に検証するデータ処理技術
10	GNSS/GPS	Global Navigation Satellite System/Global Positioning System の略。一般的に GPS を GNSS の一般名としてとらえることが多いが、GPS はアメリカの衛星測位システムであり、専門的には GNSS を衛星測位システムの一般名として用いる。
11	Image Retrieval	関係性の高そうな画像の絞り込みを行うデータ処理技術
12	Image Registration	異なる時点、異なる視点、あるいは異なるセンサから得られた二つ以上の画像を空間的に整合させる技術の総称
13	IMU	加速度角速度を取得するセンサを統合し、三次元の慣性運動を検出するモジュール
14	JPEG	Joint Photographic Experts Group の略。非可逆圧縮の画像フォーマット。DCT を用いた情報量の削減とハフマン符号化によるエントロピー圧縮で、高い圧縮率を実現する。
15	KdVisual	Kudan 社で開発された SLAM/VPS 技術。
16	LiDAR	Light Detection And Ranging の略。レーザー光を利用して対象物までの距離を計測する装置。
17	Localizer	端末のカメラなどで撮影した画像から、あらかじめ作成しておいた三次元マッ

		プデータと 対照してカメラ行列・各種パラメータなどを計算し、自己位置を推定するアプリケーショ ン処理。画像処理により特徴点を計算し対照させる手法が多い。
18	NID	Normal Information Distance の略。画像間の類似度指標の一つ
19	Renderer	3D データから 2 次元画像を生成するアプリケーション処理
20	RMS エラー	Root Mean Square Error の略で、 既知の位置と、デジタイズされた位置間の差異の尺度
21	ROS	Robot Operating System の略。ロボット・アプリケーション作成を支援するライブラリとツール
22	RTK-GNSS	Real Time Kinematic-Global Navigation Satellite System の略で、GPS など衛星を用いた「汎地球測位航法衛星システム」による測位と、地上に設置した「基準局」からの位置情報データを組み合わせることによって、高い精度の測位を実現する技術
23	SfM	Structure from Motion の略称。ある対象を撮影した複数枚の写真から、対象の形状を復元する技術の総称を指す。
24	SLAM	Simultaneous Localization and Mapping の略で、自己位置推定と環境地図 の作成を同時に行う」技術
25	SoC	System on a Chip の略。CPU や GPU などのコンポーネントを 1 チップにまとめたもの。スマートフォンで活用されている。
26	Triangulation	地点間の角度から距離を測定することで地図を作成する測量技術（三角測量）
27	Ubuntu	PC 向けの Linux OS のディストリビューションの一つ。
28	UDP	User Datagram Protocol の略。シンプルなプロトコルで到達確認をしないので、高速低遅延の通信が可能な反面信頼性は低い。
29	Unity	Unity 社が提供する 3D ゲームエンジン。
30	VPS	Visual Positioning System の略。カメラ画像と事前に得たマップ/データベースを照合して自己位置を推定する技術。

B) 五十音順

表 12-2 用語集（五十音順）

No.	用語	説明
1	カルマンフィルタ	誤差のある観測値を用いて、動的システムの状態を推定し制御するためのアルゴリズム
2	トラッキング	各時刻の自己位置を逐次的に推定すること
3	自己位置推定	自分自身の位置する座標を推測する技術。様々な方法がある。

以上

3D 都市モデルに最適化した VPS 開発 v3.0
技術検証レポート

2024 年 3 月　発行
委託者：国土交通省　都市局
受託者：TOPPAN 株式会社、株式会社ホロラボ、
　　プレティア・テクノロジーズ株式会社