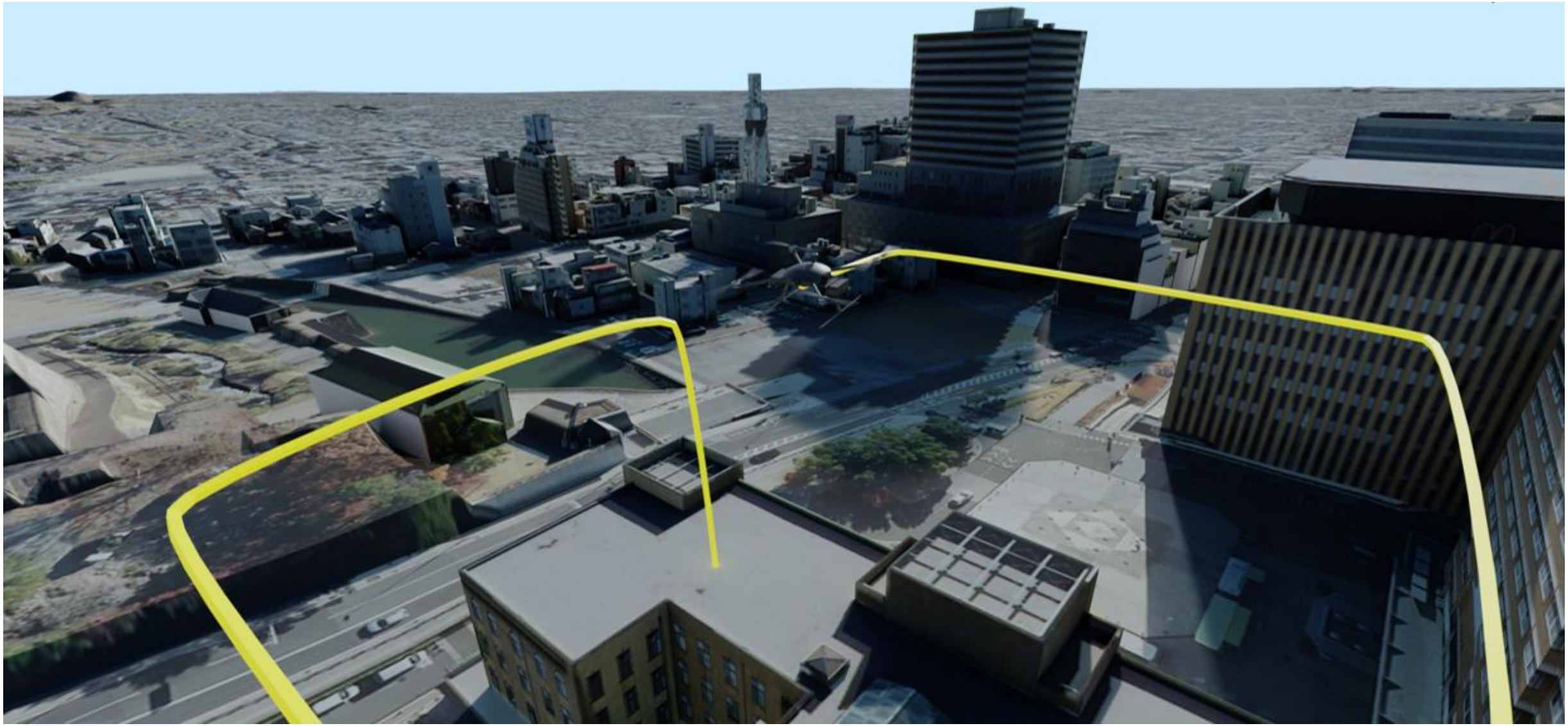
ドローンリアルタイム・ナビゲーションシステム　技術検証レポート

Technical Report for UAV real-time navigation system



目次

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．実証システム

Ⅳ．実証技術の検証

Ⅴ．成果と課題

全体概要（1/2）

ユースケース名	ドローンリアルタイム・ナビゲーションシステム
実施場所	山梨県甲府市
目標・課題 ・創出価値	• ドローンのレベル4飛行（有人地帯での補助者なし目視外飛行）の解禁に伴い、今後有人地帯での目視外飛行が一般化していくことが予想される中、ドローン同士の衝突や事故を防ぐため、飛行するドローンの高精度な自己位置推定の必要性が高まっている。 • 通常はGPSを用いた自己位置推定が一般的だが、都市部でのGPSを使った飛行はマルチパス等の影響で高い精度での自己位 • 置推定が困難なため、GPSに頼らないLiDAR SLAMやVisual SLAM技術を活用した自己位置推定の活用余地が高いと考えられる。 一方で、LiDAR SLAMやVisual SLAM技術を活用して高い自己位置推定精度を実現するには、信頼できる精度の高い事前地図が必要となるが、この地図の作成には事前の現地での撮影業務が必要となり、オペレーション上の負担が大きい。 • 今回の実証実験では、3D都市モデルの形状をマップデータとしたLiDAR SLAM及びVisual SLAMによって自己位置推定を行うシステムを開発し、3D都市モデルを3D地図としたドローン航行ナビゲーションシステムと組み合わせることで、都市部飛行における自己位置推定の精度向上やプレマップ作成にかかる時間の短縮、そして機材の軽量化（光学カメラのみでの運用も可能）など、オペレーションにおけるコストの削減を実現し、効率的かつ安全な自律飛行の実現を目指す。 さらに、自己位置推定を行う際に演算処理を機体側で行うのではなく、通信を介してサーバー上で演算処理を行う仕組みを開発する。そうすることでより柔軟性の高いシステムを構築でき、機体の性能に制限されることなく、将来的には多岐にわたる用途に活用できるドローンシステムを実現することを目指す。
ユースケース の概要	• 3D都市モデルの形状をマップデータとしたLiDAR SLAM及びVisual SLAMによって自己位置推定を行うシステムを開発する。 • 自己位置推定を行う際の演算処理を機体側ではなく、通信を介してサーバー上で演算処理を行うという期待の性能に制限されない仕組みを開発し有用性の検証を行う。

全体概要（2/2）

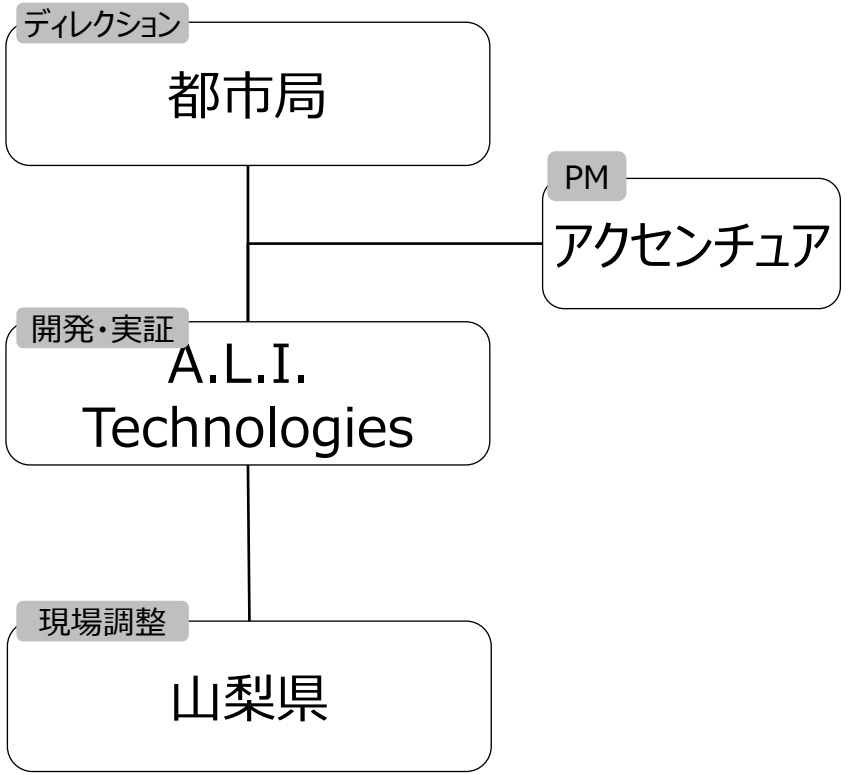
実証仮説	• 3D都市モデルを事前地図として活用することで、事前の撮影業務を行わずに精度の高い事前地図を得て、LiDAR SLAMやVisual SLAM技術による高い自己位置推定精度を実現できる。 • 上記の自己位置推定制度の高さからプレマップ作成のための事前作業等のコストが削減され、より効率的なシステムになる。 • ドローンの飛行に必要な演算処理を、機体に搭載したコンピュータではなく、ドローンの外側に準備したサーバー上で行うことで、自己位置推定精度を実現し、さらに、ストリーミング技術として有用なレベルに到達できる。 - 高精度な自己位置推定には高い演算能力が求められるが、エッジではなく通信を介してサーバー上で演算処理を行うなどの手法の確立により航行時間やペイロード（積載量）の観点からエッジ（ドローン機体上）を軽量化したいというニーズに応えられる。
検証ポイント	• 実際の飛行ルートとシステムによる推定計測ルートとを比較して、自己位置推定精度の差異を確認する。 - ベンチマーク：2023年時点で主流である位置情報のGPS（RTK）を用いた場合 - 本実証：3D都市モデルによる事前地図情報を活用したSLAM技術の場合 • ドローンの機体上に設置したセンサーからのデータ通信速度を3種類のネットワーキング構成（LTE、5G、LAN）から比較して、自己位置推定の演算処理速度の差異を確認する。 - ベンチマーク：エッジ（ドローン機体上）で演算処理を行った場合 - 本実証：ドローンの外側に準備したサーバーで演算処理を行った場合

実施体制

各主体の役割

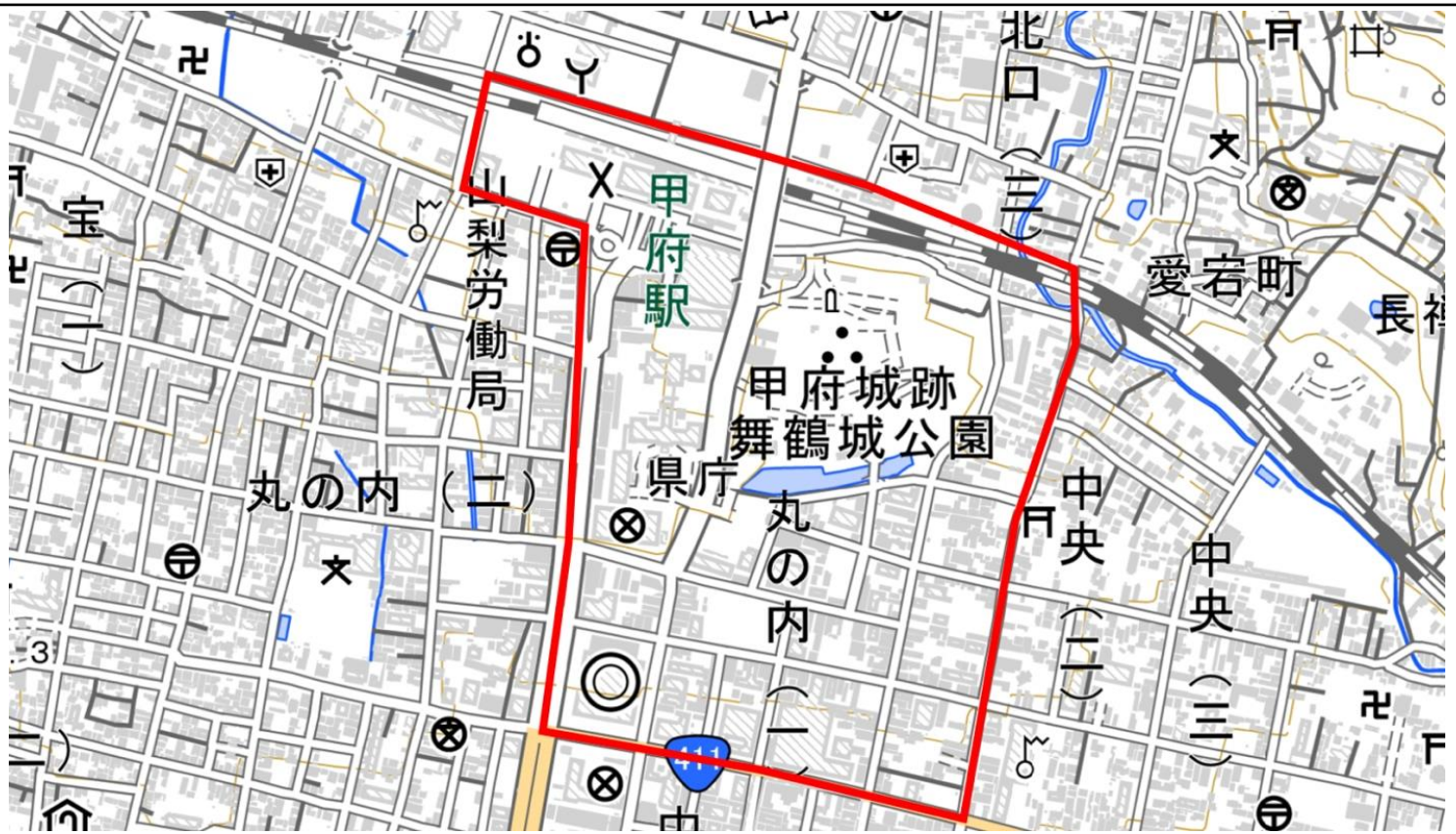
主体	役割
A.L.I. Technologies	• システム開発 • 実証飛行実施 • 報告書の作成
山梨県	• 実証フィールドの提供 • 自治体側調整業務
アクセンチュア	• プロジェクトマネジメント

実施体制図



実証エリア

山梨県 甲府市 山梨県庁周辺地域　1.95km^2



スケジュール

実施事項	令和4年										令和5年		
	3月	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. 運航管理サーバー開発				←	—	→							
2. 機体ハードウェア開発				←	—	—	→						
3. LiDAR SLAMシステム開発				←	—	—	→						
4. LiDAR SLAMシステム試験							←	→					
5. Visual SLAMシステム開発						←	—	—	—	—	→		
6. Visual SLAMシステム試験											←	→	
7. Shared Computingサーバーセットアップ							←	—	—	—	→		
8. 実証試験											←	→	
9. 報告書作成、成果報告											←	—	→

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．実証システム

Ⅳ．実証技術の検証

Ⅴ．成果と課題

活用技術一覧

本実証に用いる技術は下記の通り

項目	内容
LiDARセンサー	• 光学レーザーを用いて周辺環境の3D測定を行うことが可能なセンサー
ステレオカメラ	• 2つのカメラを使い対象物を複数の異なる方向から同時に撮影することにより、人の眼と同じような仕組みで奥行き情報を取得することが可能なカメラ
SLAM	• センサー等を用いて空間の中で自己位置推定と環境地図作成を同時に行う技術
ROS	• ロボットのアプリケーション開発を支援するライブラリやツール群 • ハードウェア抽象化、センサーやアクチュエータ等のデバイスドライバ、ロボットに搭載される汎用機能の実装、機能同士のデータ通信などの機能を有する
VPN	• VPNはインターネットを介した接続されたマシン・ネットワークに対して仮想的なローカルネットワークを構築する技術であり、物理的に離れたマシン同士があたかも同一のローカルネットワークにあるかのように通信を行うことが可能
COSMOS	• A.L.I. Technologiesが開発するドローン管制システム • ドローンの機体やオペレータ、飛行ログの管理や飛行状況のライブモニタリングなどの機能を有する
Unity	• ゲーム開発やシミュレーション、仮想現実、拡張現実などのアプリケーション開発に使用される、クロスプラットフォームの統合開発環境
Blender	• 3Dモデリング、アニメーション、シミュレーション、レンダリング、ビデオ編集を含む広範な機能を提供する。オープンソースの3Dコンピューターグラフィックスソフトウェア

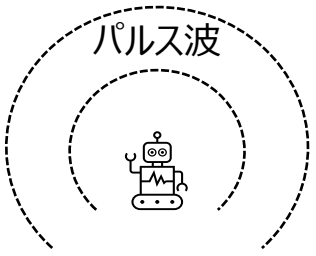
LiDARセンサーについて

光学レーザーを用いて周辺環境の3D測定を行うことが可能なセンサー

LiDARセンサーの動作原理*

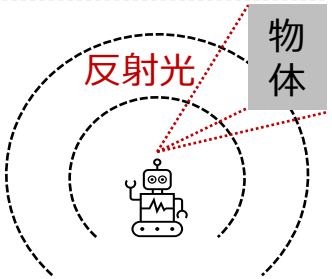
周辺にパルス波を照射

- 3D空間上にパルス波という無数のレーザーを照射する



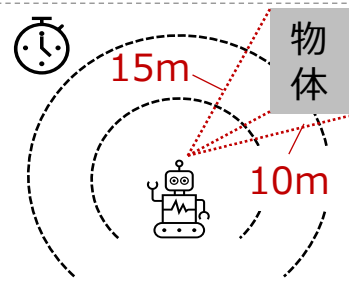
物体からの反射波を検出

- 発射したレーザーが周辺の物体によって跳ね返ってくる反射光を検出する



物体までの距離を算出

- 反射光が返ってくるまでの時間から物体までの距離を算出することで周辺環境の把握を行う
- 本プロセスを連続的に行うことで、ドローンの周辺環境をリアルタイムでセンシングすることができる



*: Velodyne社の公式サイトに掲載されている概念図を参考に作成
Source: https://velodynelidar.com/what-is-lidar/

代表的なLiDARセンサー

- Ouster社 OS1-128
  - 回転式LiDAR（全方位的な点群取得が可能）
  - 最大範囲200m、垂直視野45°

- Velodyne社 Velarray M1600
  - ソリッドステート式LiDAR（軽量・安価、正面のみの点群取得）
  - 最大範囲30m、水平視野120°、垂直視野32°

ステレオカメラについて

2つのカメラを使い対象物を複数の異なる方向から同時に撮影することにより、人の眼と同じような仕組みで奥行き情報を取得することが可能なカメラ

概要

項目	内容
名称	ステレオカメラ
概要	• 対象物を複数の異なる方向から同時に撮影することにより、その奥行き方向の情報も記録できるようにしたカメラ • 通常のカメラのRGB（可視光）情報に対し、奥行情報を加えたRGB-D（D：Depth）情報を取得できる
仕組み	• 2つのカメラの偏差を利用して奥行情報を算出 - 人の眼と同じ仕組みで眼となるカメラを一定の間隔をあけて設置することで生じる視差から奥行情報を推定する

代表的なステレオカメラ

- Intel社 RealSense d435i
  - 出力解像度
    1920 x 1080 @30fps
  - 画角
    69°(H) x 42°(V)
  - 深度精度
    < 2% up to 2m

- STEREOLAB社 ZED i2
  - 出力解像度
    1920 x 1080@30fps
  - 画角
    110°(H)x70°(V)x120°(D)
  - 深度精度
    < 1% up to 3m
    < 5% up to 15m

SLAMについて

センサー等を用いて空間の中で自己位置推定と環境地図作成を同時に行う技術

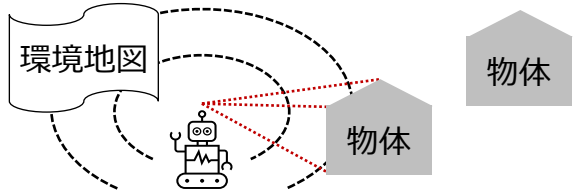
概要

項目	内容
名称	SLAM（Simultaneous Localization and Mapping）
概要	• SLAMとは、ある空間の中で空間内を走査するセンサーを使い、自己位置推定と環境地図作成を同時に行う仕組み - 自己位置推定：空間の中で自身がどこにいるかを空間地図とセンサーデータを突合し推測 - 環境地図作成：センサーのデータから空間の地図を作成
分類	• センサー別（代表例） - LiDAR SLAM：LiDARセンサーを利用 - Visual SLAM：カメラを利用 • 事前地図の有無 - 環境地図をあらかじめ保有する - 環境地図を０から作りながら自己位置推定も行う

事前地図の有無による違い

①事前地図なし

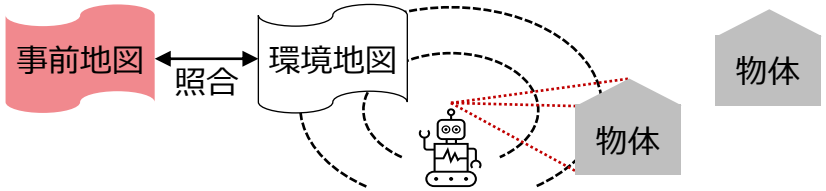
センサーデータからリアルタイムに環境地図を作成して自己位置を推定



- 搭載したセンサーや演算によって生じる誤差の影響が相対的に大きくなる
- 例：家庭用掃除ロボット

②事前地図あり

事前地図とセンサーデータに基づく環境地図を照合させながら自己位置を推定



- 事前地図の精度が高い場合、①よりも自己位置推定の精度が高くなることが期待できる
- 都市部のように広範囲かつ複雑な形状である場合、精度の高い周辺環境地図データが得られないこともあるため、用途や目的により使い分けが必要

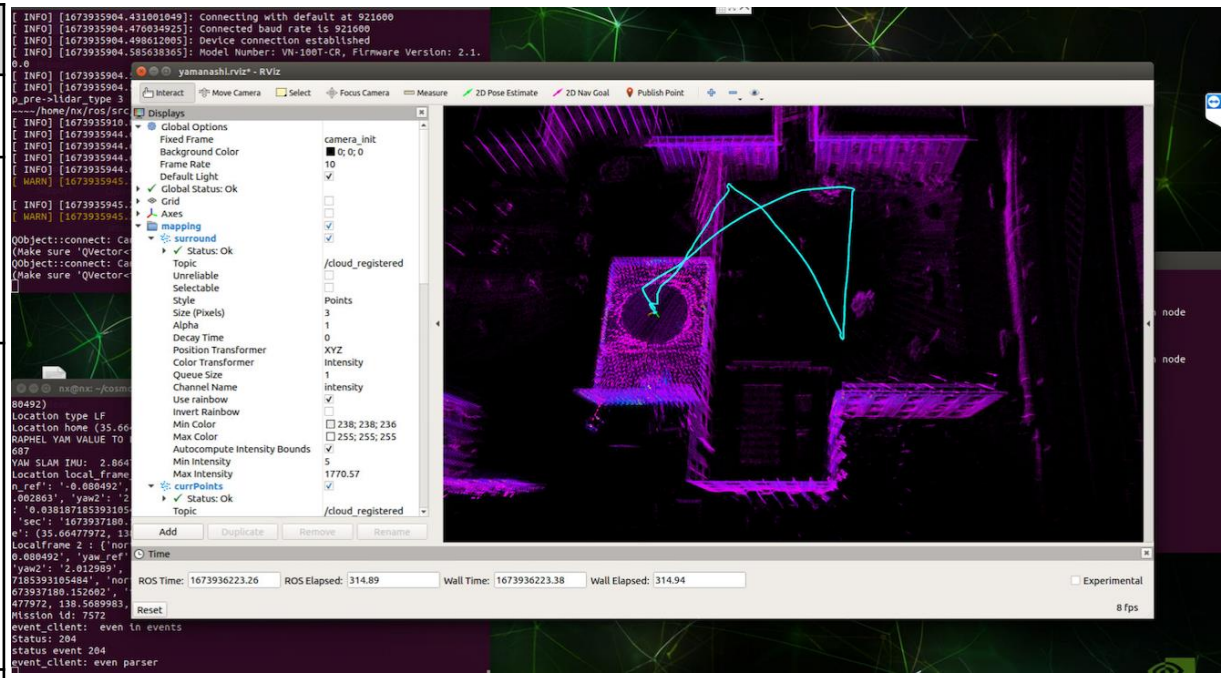
ROSについて

ロボットのアプリケーション開発を支援するライブラリやツール群であり、ハードウェア抽象化、センサーやアクチュエータ等のデバイスドライバ、ロボットに搭載される汎用機能の実装、機能同士のデータ通信などの機能を有する

ROSの概要

項目	内容
名称	ROS （Robot Operating System）
概要	• 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のパッケージ • オープンソースで提供
主な機能	• ハードウェア抽象化 • デバイスドライバ • ライブラリ • 視覚化ツール • メッセージ通信 • パッケージ管理、など
利用する機能	• センサー類を利用するためのデバイスドライバ • 各データのやり取り • 視覚化ツール（Rviz）

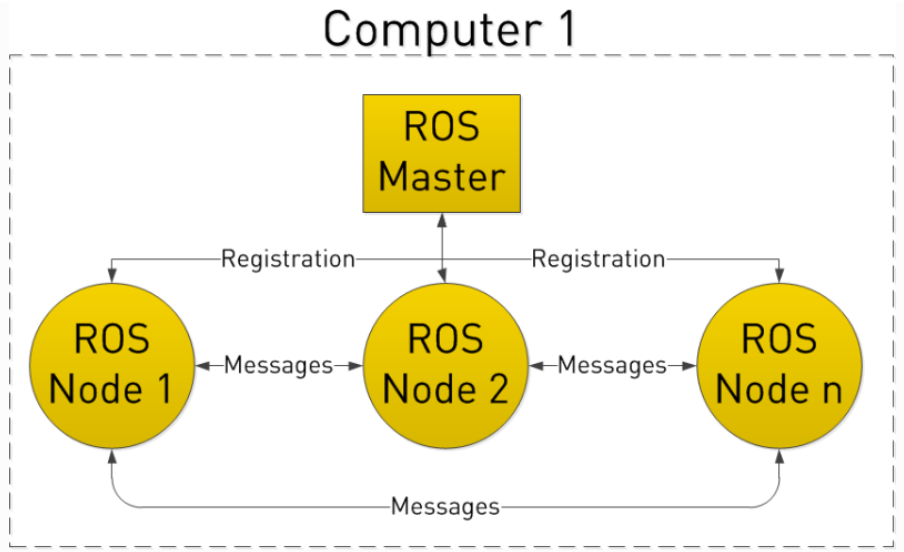
ROSベース視覚化ツールRvizの画面イメージ



ROSの構成イメージ

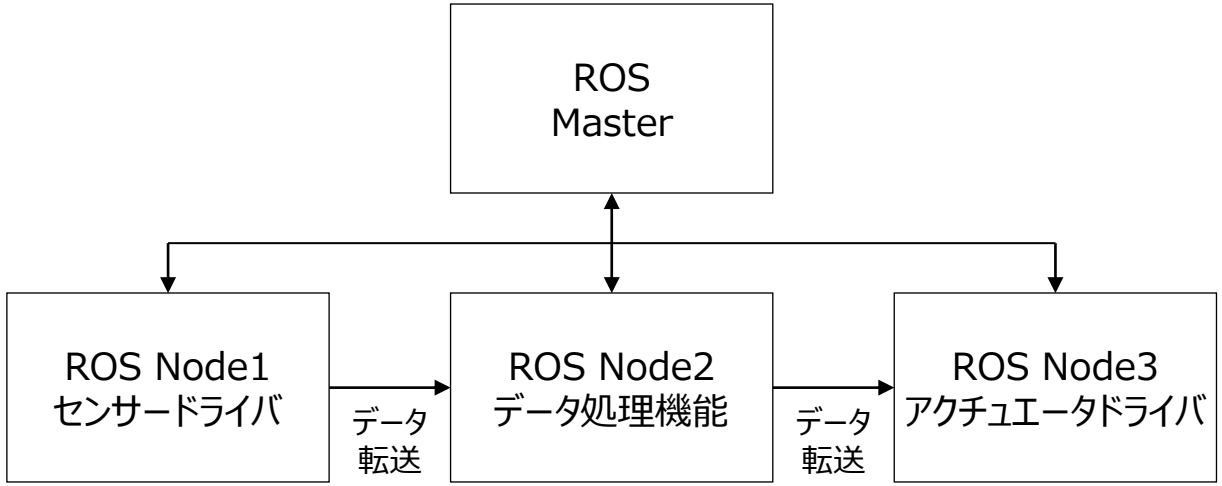
ROSは機能を担うNodeとNodeを統括するMasterで構成されており、この構造によってさまざまなセンサーやアクチュエータを扱うロボットアプリケーションに対応可能

ROSの構成イメージ（ROS Graph）



- ROSの基本構成要素
  - ROS Node（ノード）：各Nodeは異なる機能を有する
  - ROS Master（マスター）：多数のNodeを統括する
- Node-Node間は、様々なトピックやサービスの形でデータ通信されている
- 適切なNode構成とパラメータ設定により、様々なロボットアプリケーションの実装が可能となる

ROSのシンプルな実装例



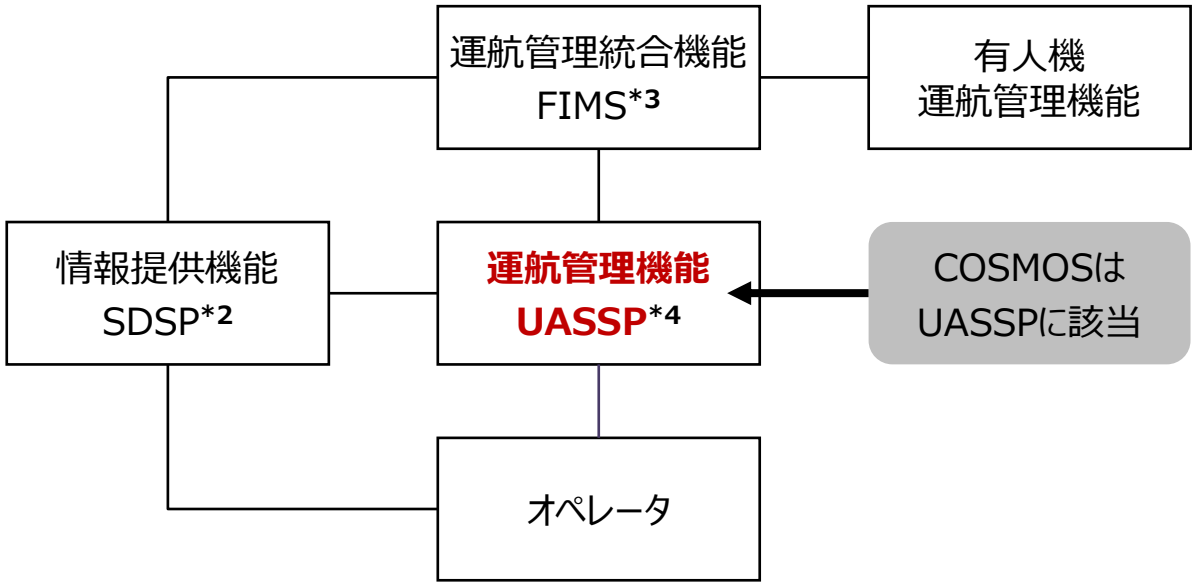
- 各ROS Nodeの機能
  - センサードライバ：センサーでデータを取得
  - データ処理機能：センサーデータの演算処理を行う
  - アクチュエータドライバ：演算処理結果をもとにロボットを動かす

Source: Clearpath社公式サイト（2023年2月）（https://www.clearpathrobotics.com/assets/guides/melodic/ros/Intro）

COSMOSについて

A.L.I. Technologiesが開発するドローン管制システムであり、ドローンの機体やオペレータ、飛行ログの管理や飛行状況のライブモニタリングなどの機能を有する

無人航空機運航管理システムアーキテクチャ*1におけるCOSOMOの位置づけ



COSMOSの特徴

- 複数台のドローンの飛行を同時管理
  - 複数台のドローン飛行を一元的に管理できる技術（特許取得済）を実装
- 安心安全を可視化する技術
  - フライト航路の情報、飛行する機体情報、オペレータ情報、機体情報など、ドローンの飛行管理に必要となるあらゆる情報を一元的に集約
  - GPS及び上空LTEを利用して補足された現在位置を地図データと3D空間上に投影し、自治体や運航事業者でモニタリングすることにより、安全に運航しているかを可視化する可能
- 遠隔操作でオペレーションをバックアップ
  - ドローンの操縦者が不測の事態でオペレーションが継続できない場合や、甚大な広域災害が発生し被災地でオペレーションが完結できない場合を想定し、遠隔操作によるバックアップを実現

*1Source: 新エネルギー・産業技術総合開発機構　DRESSプロジェクト運航管理システムを使ったドローン運航ビジネスの姿（www.nedo-dress.jp/wp-content/uploads/2022/02/運航管理システムを使ったドローン運航ビジネスの姿.pdf）
*2: Supplementary Data Service Providerの略称で、FIMSやオペレータに対してドローンの運航を補助するための情報を提供する
*3: Flight Information Management Systemの略称で、複数のUASSPから送信される情報を統合し、各UASSPに情報を送信する
*4: Unmanned Aircraft System Service Providerの略称で、オペレータに他紙してドローンの運航に必要となる機能や情報を提供する

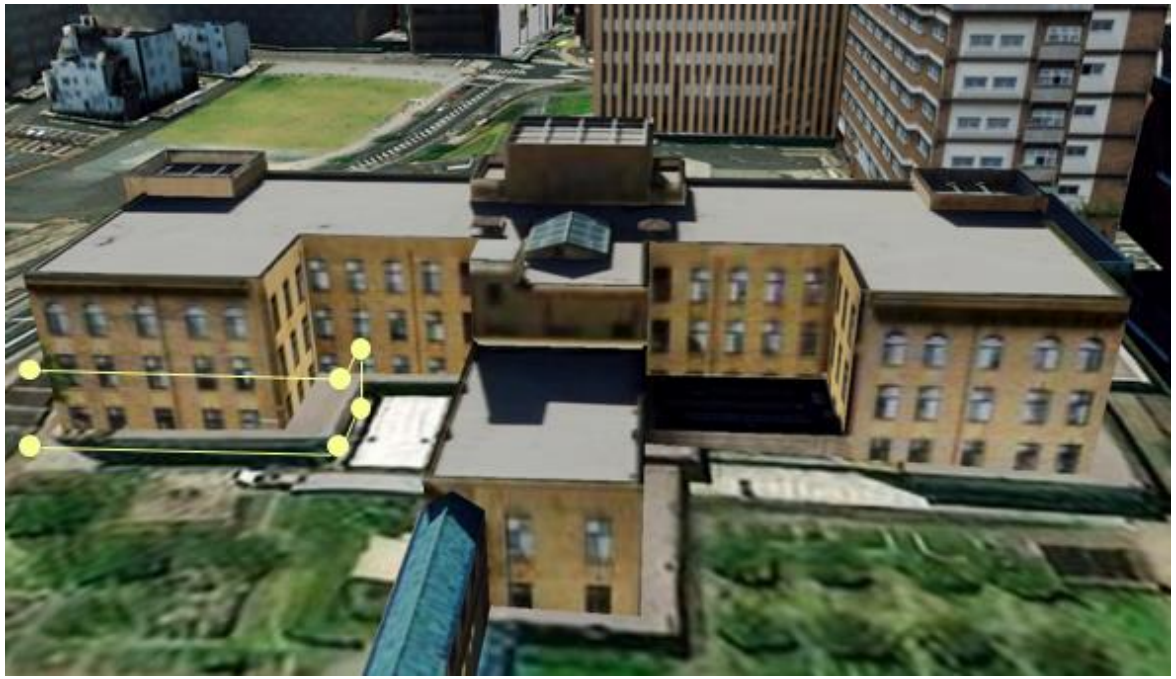
Unityについて

Unityは直感的なツール開発や、現実世界の物理法則に合わせた挙動を制御・操作できる物理エンジンを有するソフトウェア

概要

項目	詳細
名称	Unity
概要	• 直感的なゲーム開発が可能なゲームエンジン • 「Unity Script」と呼ばれる拡張機能により独自の処理を実装可能 • アプリケーションの開発も可能
主な機能	• レンダリング • スクリプトによる機能拡張 • アセットの取り込み、編集、作成 • 仮想カメラによる空間の映像取り込み
本ユースケースで利用する機能	• 仮想カメラによるATLASMAP生成

Unityに取り込んだ3D都市モデルを仮想カメラで撮影



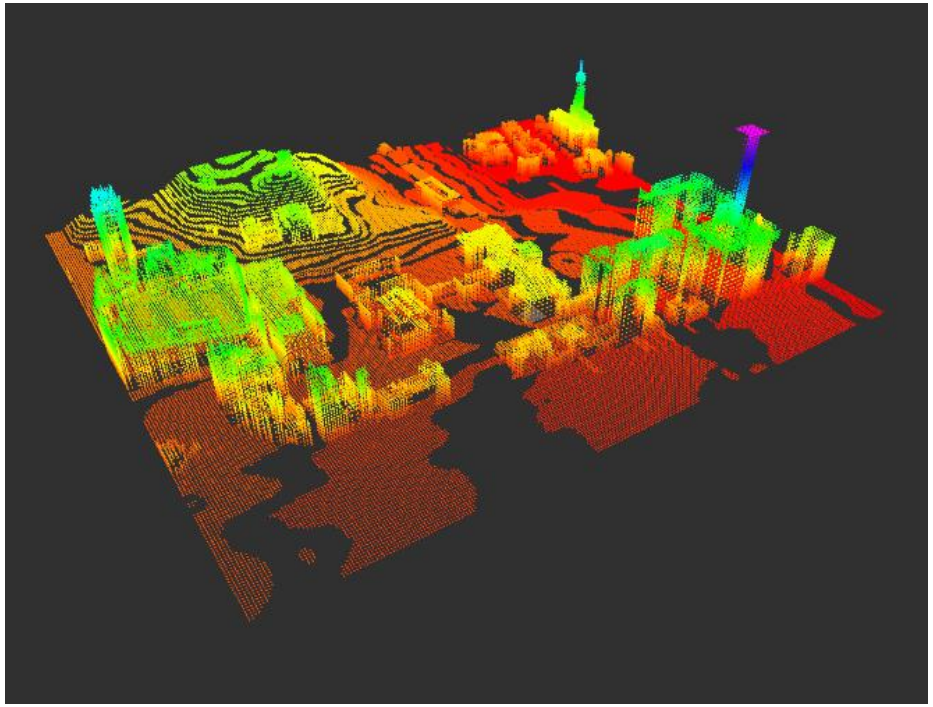
Blenderについて

オープンソースの統合型3DCG制作ソフトで、BlenderGISアドオンを使うことでGISデータにも対応する

概要

項目	詳細
名称	Blender
概要	• 3DCGアニメーションを作成するためのオープンソースの統合環境アプリケーション • 3Dのモデリング、レンダリング、アニメーション等幅広い用途に利用可能
主な機能	• 3Dモデリング（作成、編集、三角メッシュ化等） • 3Dレンダリング • データ変換
本ユースケースで利用する機能	• Octomapの生成

BlenderによるOctomap生成イメージ



※1 https://www.blender.jp

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．実証システム

Ⅳ．実証技術の検証

Ⅴ．成果と課題

実証フロー

開発したハードウェア・ソフトウェアを装備したドローンの飛行試験を実施して、各種データを取得した後、システムの精度検証と有用性評価を行う

要件定義	• 本実証実験を行うために必要なハードウェアの選定や、既存システムとの接続、新規開発が必要な機能・モジュールの整理を行う
システム開発	• 各コンポーネント（機体側・サーバー側）に必要な機能・システムの開発を行う • ドローンに各種必要な機材を搭載した状態で飛行試験を実施し、飛行性能の確認を行う
実証実験	• 山梨県甲府市にて実際にドローン飛行を実施し、各種データの取得を行う
精度検証・ 有用性評価	• 実証実験で得られたデータをもとに解析を行い、自己位置推定の精度検証や新規開発したシステムの有用性評価を行う

実証シナリオ

項目	①LiDAR SLAMエッジ処理	②LiDAR SLAMサーバー処理	③Visual SLAMエッジ処理	④Visual SLAMサーバー処理
実施内容	• ドローンに搭載されたLiDARセンサーから得られる点群データを用い、自己位置推定のための演算処理を機体側（エッジ側）で行う • この処理は、機体に搭載されている小型のコンパニオンコンピュータであるJetson Xavier NXで行われる	• LiDARセンサーから得られる点群データを外部のサーバー上へ伝達し、サーバー上で自己位置推定のための演算処理を行う • 飛行中は、LTEや5G等の通信方法を使用し、サーバーとのデータのやり取りを行う	• ドローンに搭載されているステレオカメラから得られるデータを用い、自己位置推定のための演算処理を機体側（エッジ側）で行う • この処理は、ドローンに搭載されている小型のコンパニオンコンピュータであるJetson Xavier NXで行われる	• Visual SLAMから得られる情報を外部のサーバー上へ伝達し、サーバー上で自己位置推定のための演算処理を行う • 飛行中は、LTEや5G等の通信方法を使用し、サーバーとのデータのやり取りを行う
3D都市モデルの主な利用方法	• 3D都市モデルをLiDAR SLAMの事前地図として活用するために、Octomapライブラリを利用してグリッドマップを作成し、Jetsonへ事前地図データとしてプリロードする • 飛行中には、3D都市モデルから生成された事前地図とセンサーから得られる点群をマッチングし、リアルタイムに自己位置推定を行う • 3D都市モデルは、COSMOSサーバーへもプリロードされ、運航管理システムの3D空間上で可視化される	• 3D都市モデルをLiDAR SLAMの事前地図として活用するために、Octomapライブラリを利用してグリッドマップを作成し、シェアドコンピューティングサーバーへ事前地図データとしてプリロードする。 • 飛行中には、3D都市モデルから生成された事前地図とセンサーから得られる点群をマッチングし、通信を介しながらリアルタイムにサーバー上で自己位置推定を行う • 3D都市モデルは、COSMOSサーバーへもプリロードされ、運航管理システムの3D空間上で可視化される	• 事前にUnity上で、3D都市モデルを配置したシミュレータを構築し、仮想カメラを使った模擬飛行を行うことで、3D都市モデルを元にした事前地図（ATLAS MAP形式）を作成する • 事前地図は、エッジ側のJetsonへ保存される • 飛行中には、3D都市モデルから生成された事前地図とセンサーから得られる特徴点をマッチングし、リアルタイムに自己位置推定を行う • 3D都市モデルは、COSMOSサーバーへもプリロードされ、運航管理システムの3D空間上で可視化される	• 事前にUnity上で、3D都市モデルを配置したシミュレータを構築し、仮想カメラを使った仮想飛行を行うことで、3D都市モデルを元にした事前地図（ATLAS MAP形式）を作成する • 事前地図は、シェアドコンピューティングサーバーへ保存される • 飛行中には、3D都市モデルから生成された事前地図とセンサーから得られる特徴点をマッチングし、通信を介しながらリアルタイムにサーバー上で自己位置推定を行う • 3D都市モデルは、COSMOSサーバーへもプリロードされ、運航管理システムの3D空間上で可視化される

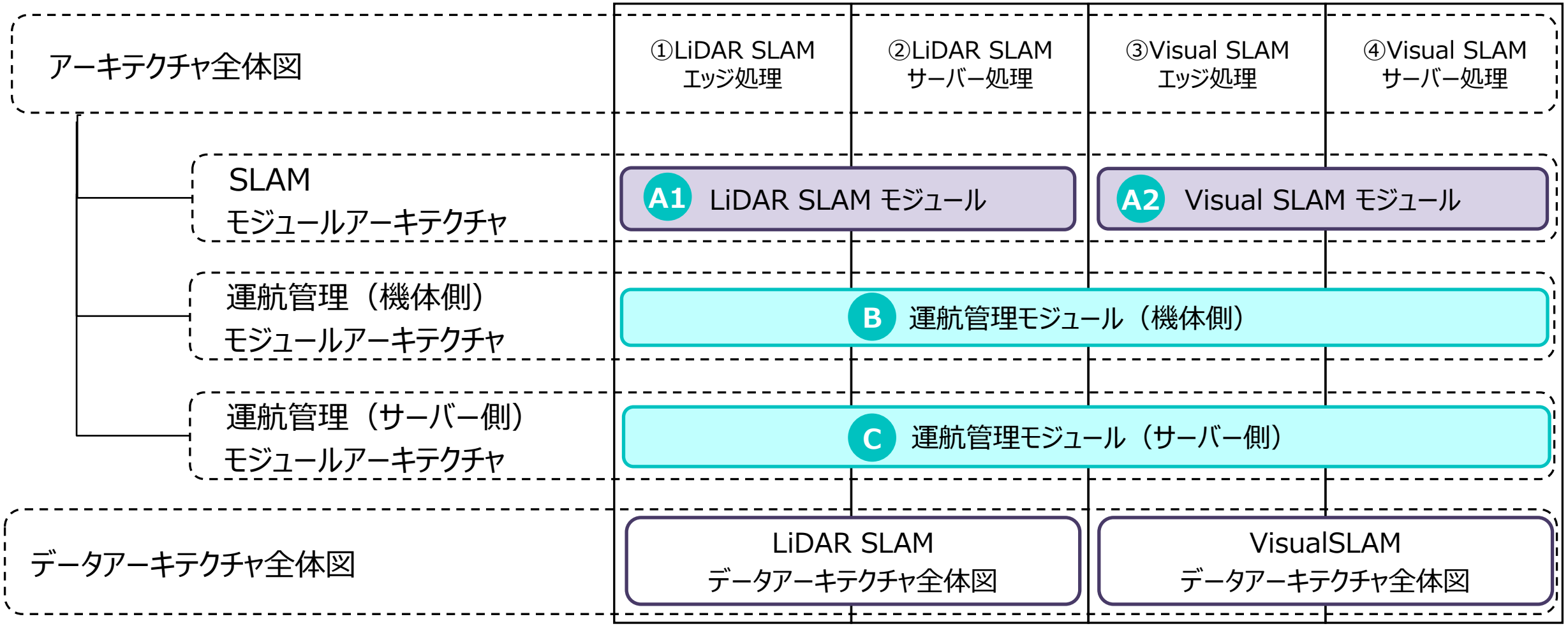
想定事業機会

ドローン管制システムのCOSMOSを起点とした自治体やドローンを利用・製造する民間事業者向けのビジネスモデル・マネタイズを想定

項目	内容
利用事業者	• 自治体 • 物流事業者 • ドローン製造業者 • その他ドローンを活用する事業者
提供価値	• 3D都市モデルに対応した安全で精度の高いドローンの運航 • COSMOSを介した遠隔からの運航状況の把握、可視化 • 各社ドローンへの組み込みおよび技術サポート
サービス仮説	• オペレータ・機体の提供までを含めたドローン運航サービス（Drone as a Service） - 物流事業者等の業務の省力化につながるドローン運航サービスを提供する • 飛行ログ提供サービス - ドローンの墜落や事故の際に、第三者として状況に対する客観的な情報提供を行う

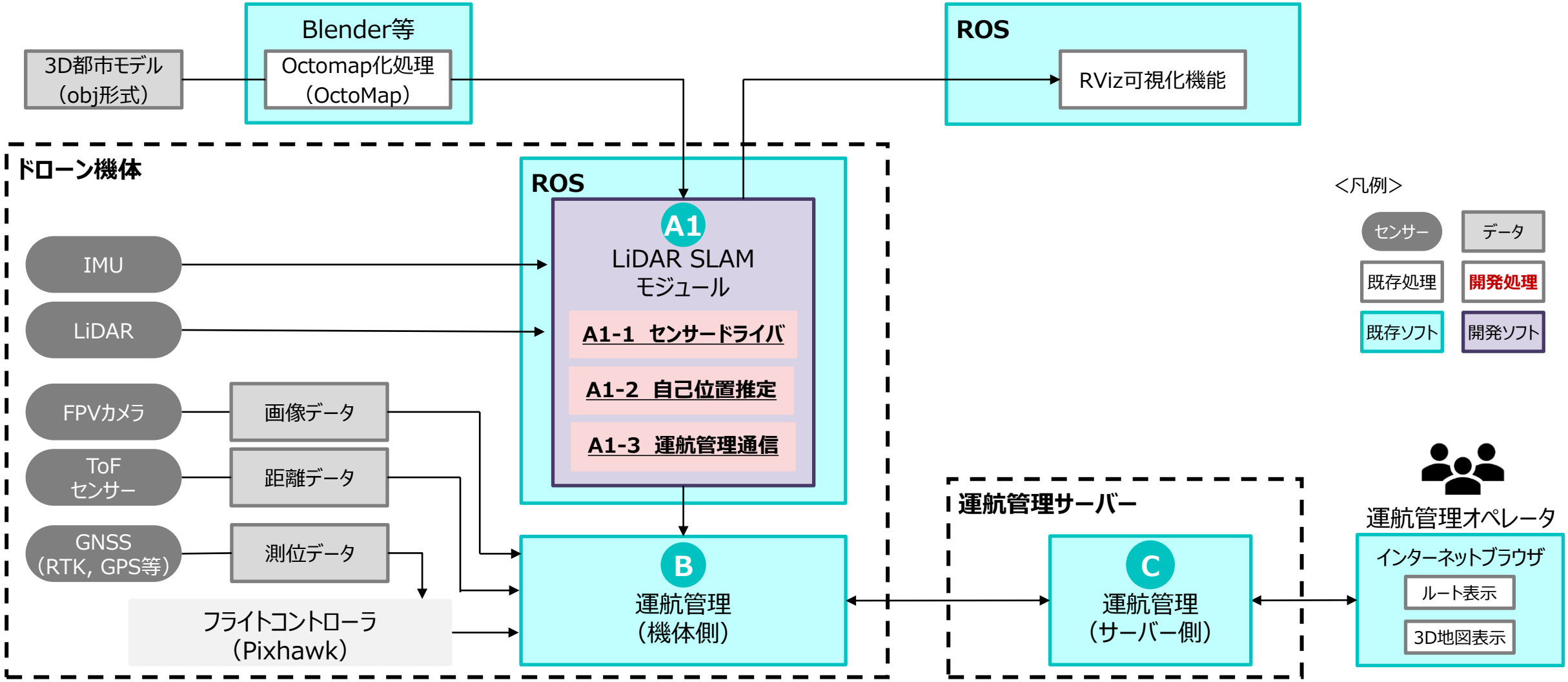
システムアーキテクチャ

本実証にて開発したシステムのアーキテクチャを本章にて示す。各実証シナリオと各アーキテクチャ全体図およびモジュールの関係を下表に示すとおりである。



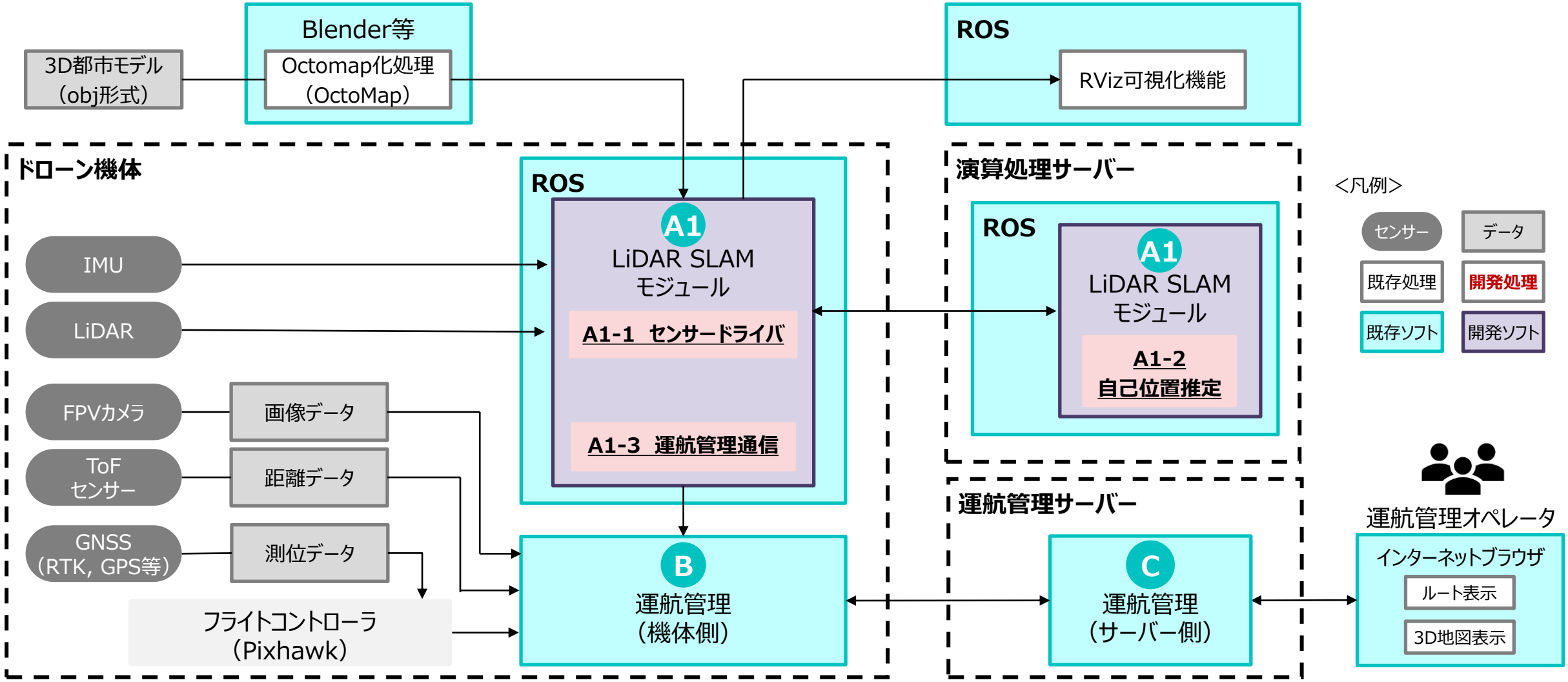
アーキテクチャ全体図（①LiDAR SLAMエッジ処理）

LiDAR SLAMエッジ処理のアーキテクチャ全体図は下記の通り



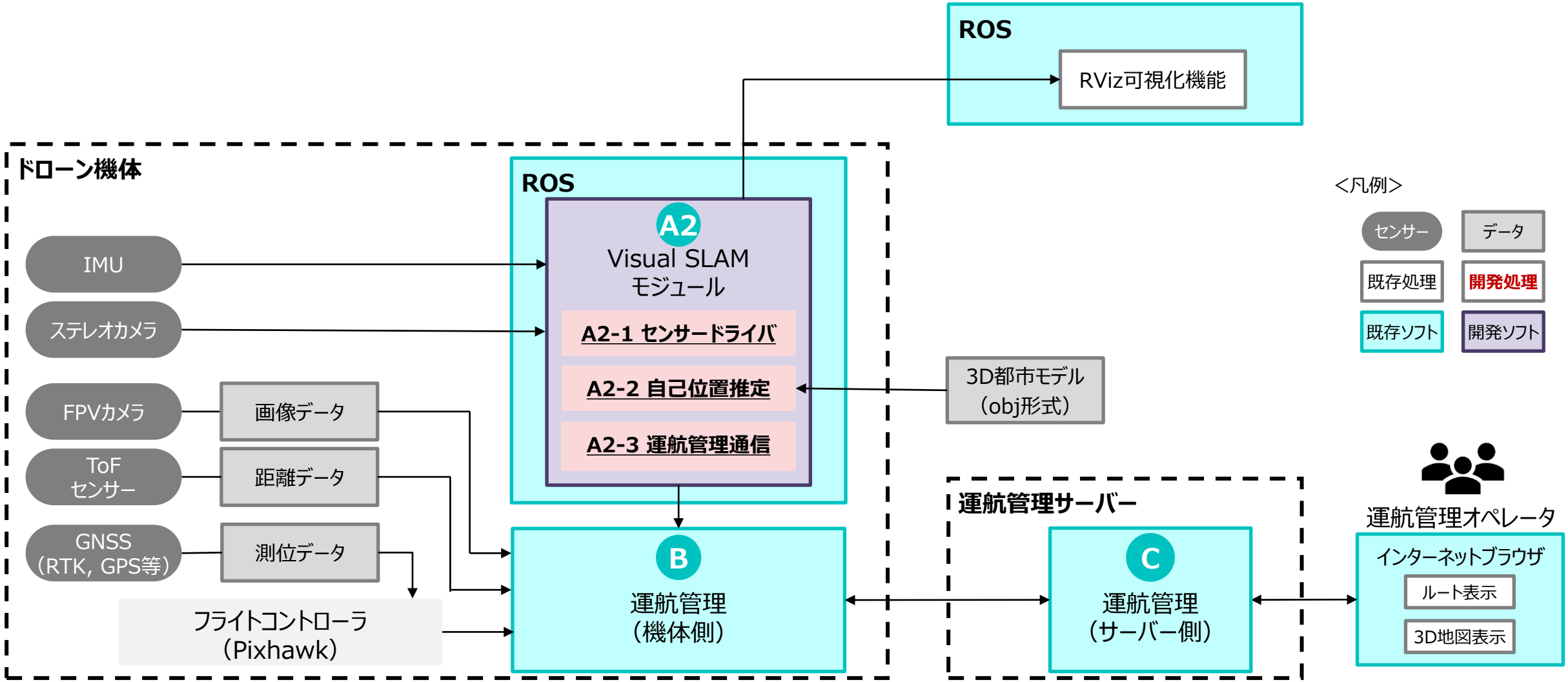
アーキテクチャ全体図（②LiDAR SLAMサーバー処理）

LiDAR SLAMサーバー処理のアーキテクチャ全体図は下記の通り



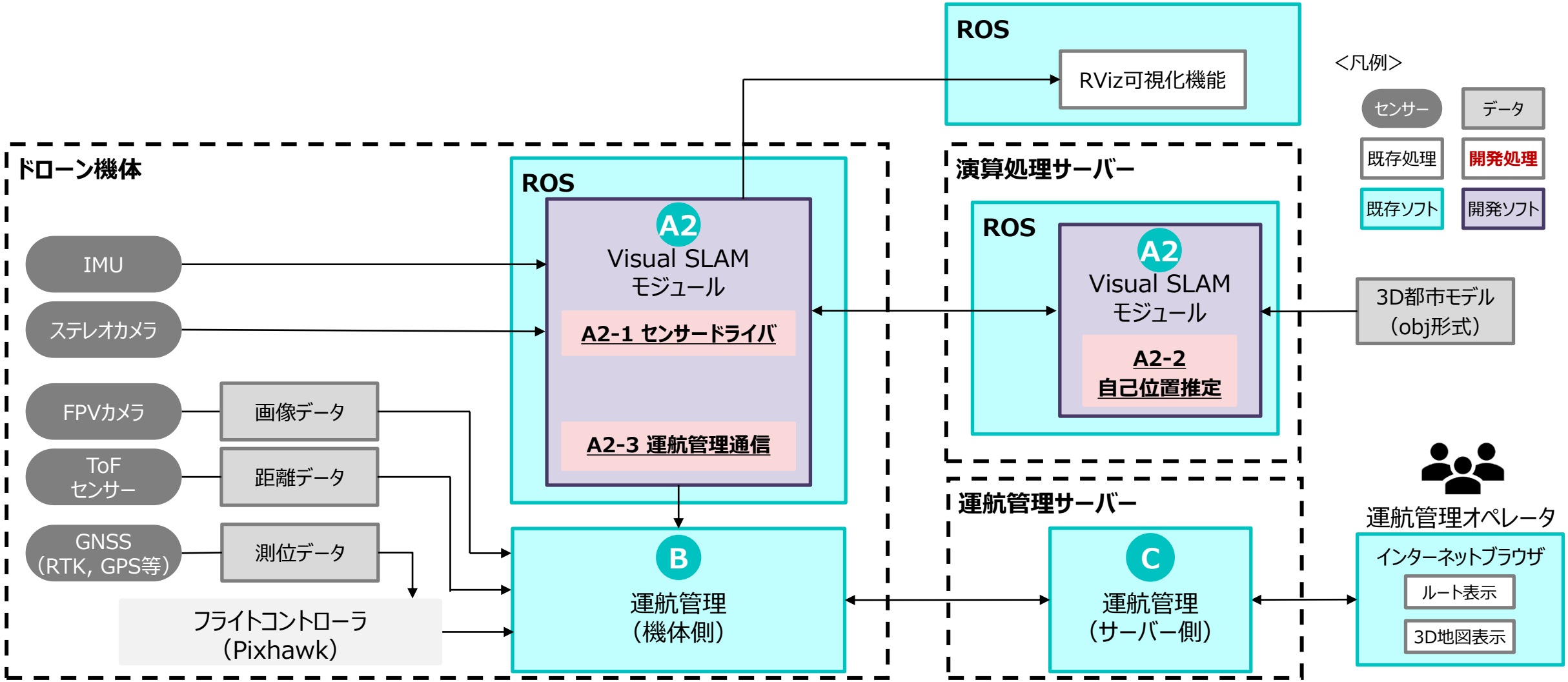
アーキテクチャ全体図（③Visual SLAMエッジ処理）

Visual SLAMエッジ処理のアーキテクチャ全体図は下記の通り



アーキテクチャ全体図（④Visual SLAMサーバー処理）

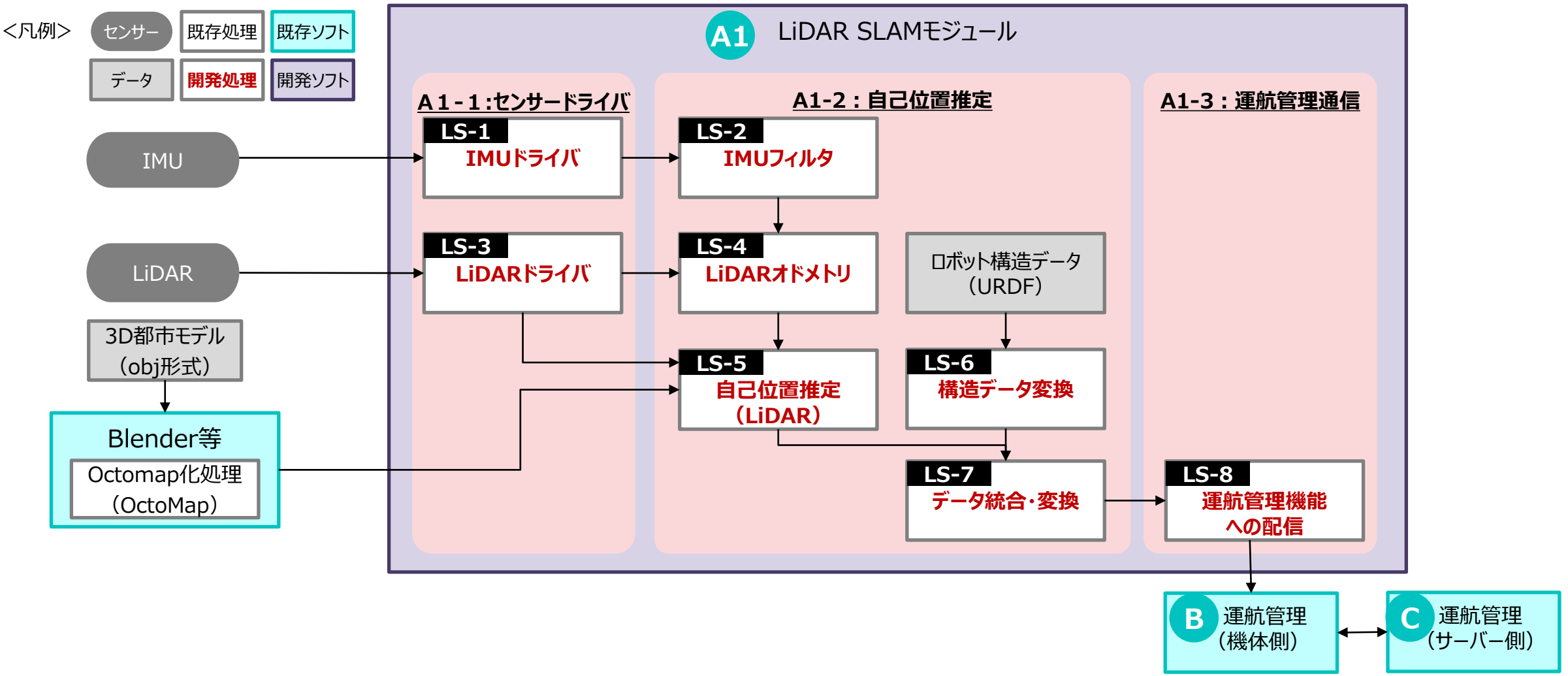
Visual SLAMサーバー処理のアーキテクチャ全体図は下記の通り



Ⅲ. 実証システム ＞ 3. アーキテクチャ全体図

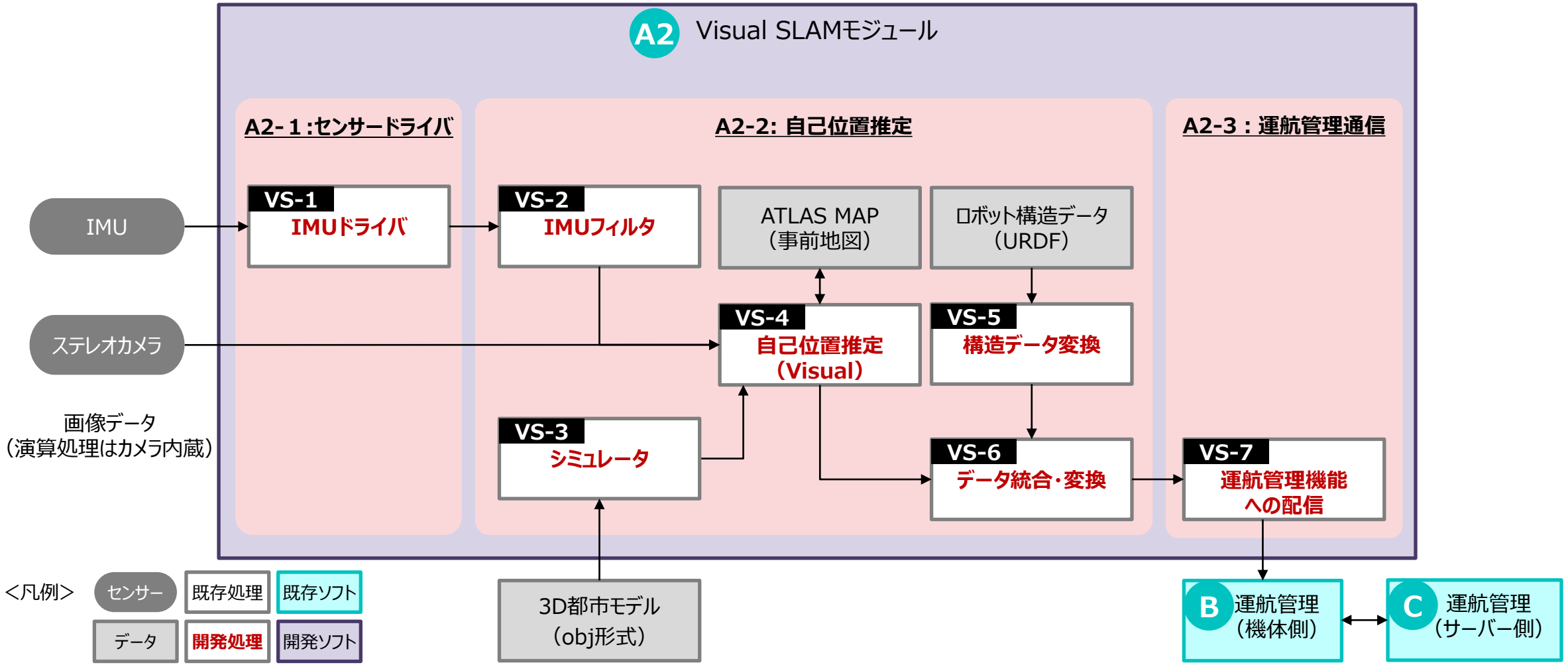
モジュールアーキテクチャ | A1 LiDAR SLAMモジュール

LiDAR SLAMのモジュールアーキテクチャ全体図は下記の通り



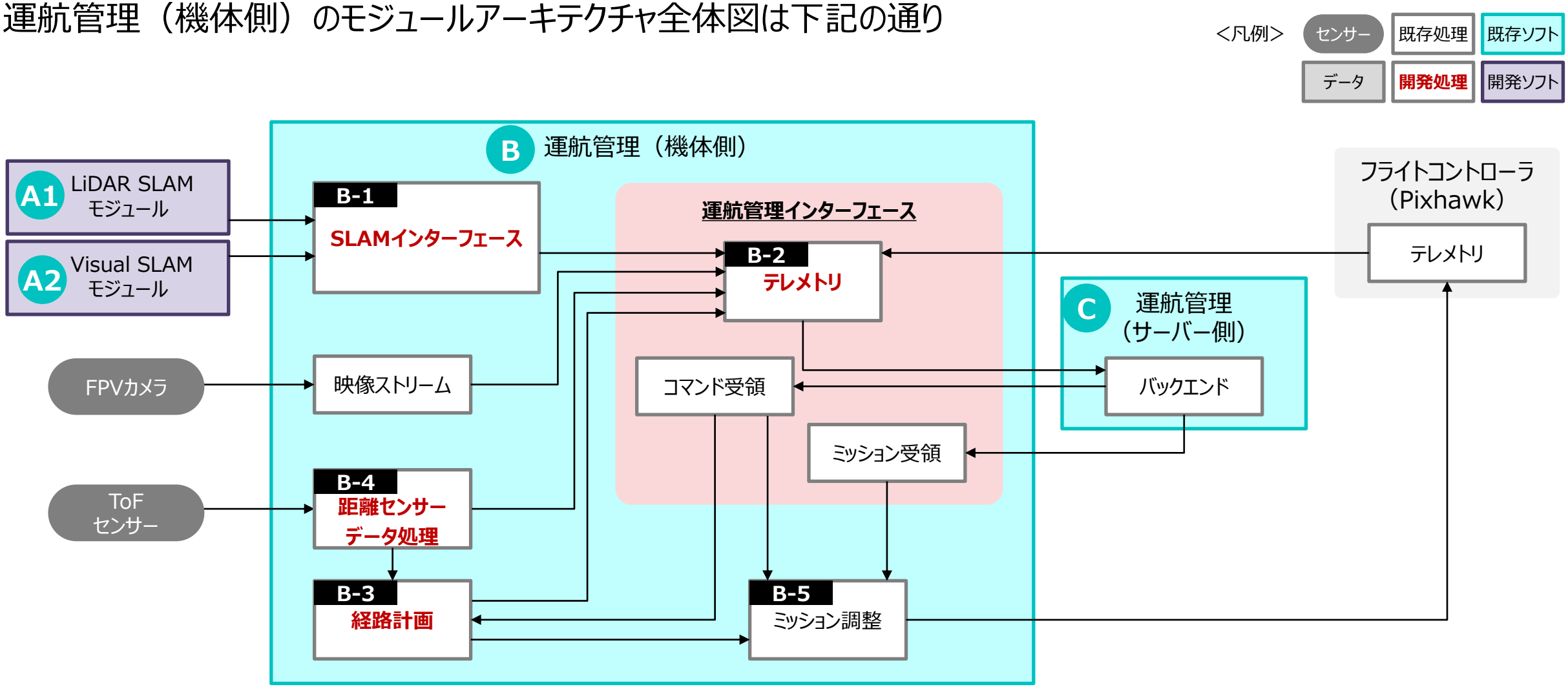
モジュールアーキテクチャ｜A2 Visual SLAMモジュール

Visual SLAMのモジュールアーキテクチャ全体図は下記の通り



モジュールアーキテクチャ｜B 運行管理（機体側）

運航管理（機体側）のモジュールアーキテクチャ全体図は下記の通り



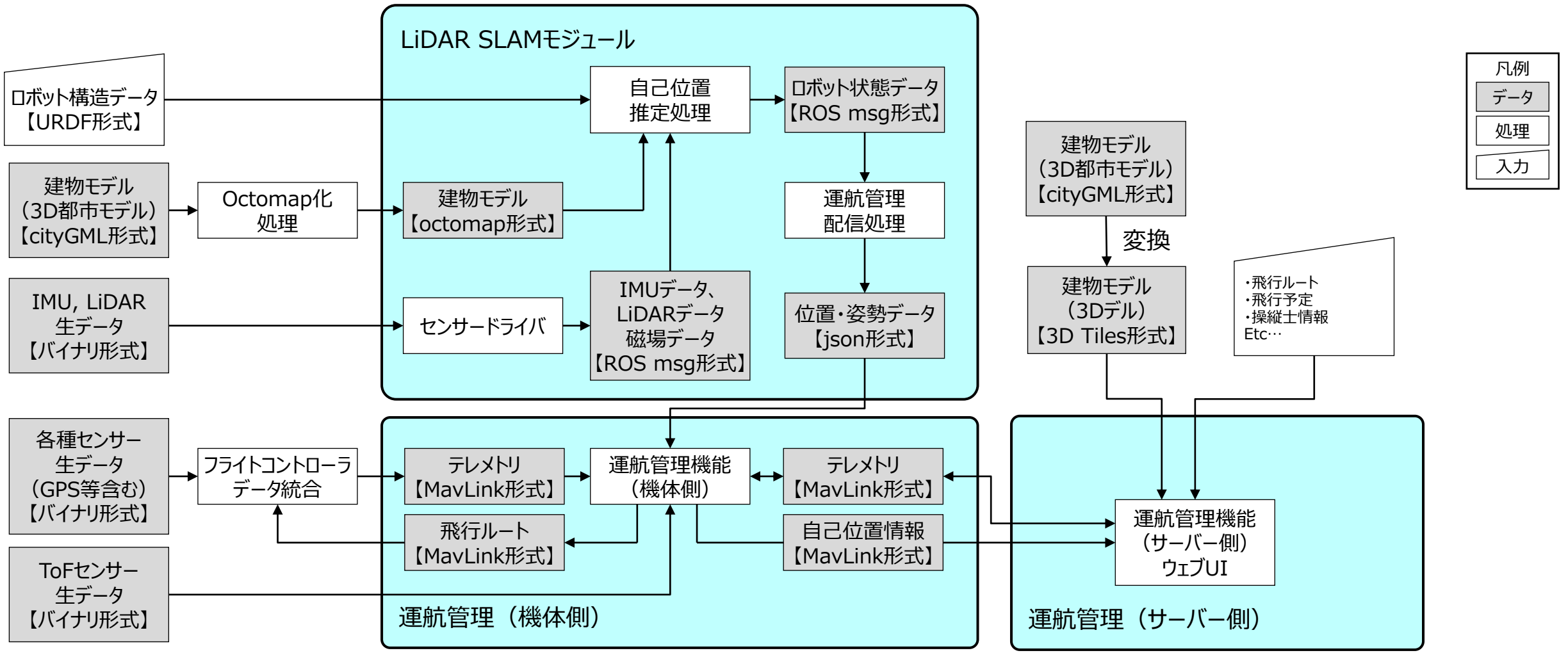
Ⅲ. 実証システム ＞ 3. アーキテクチャ全体図

モジュールアーキテクチャ｜C運行管理（サーバー側）

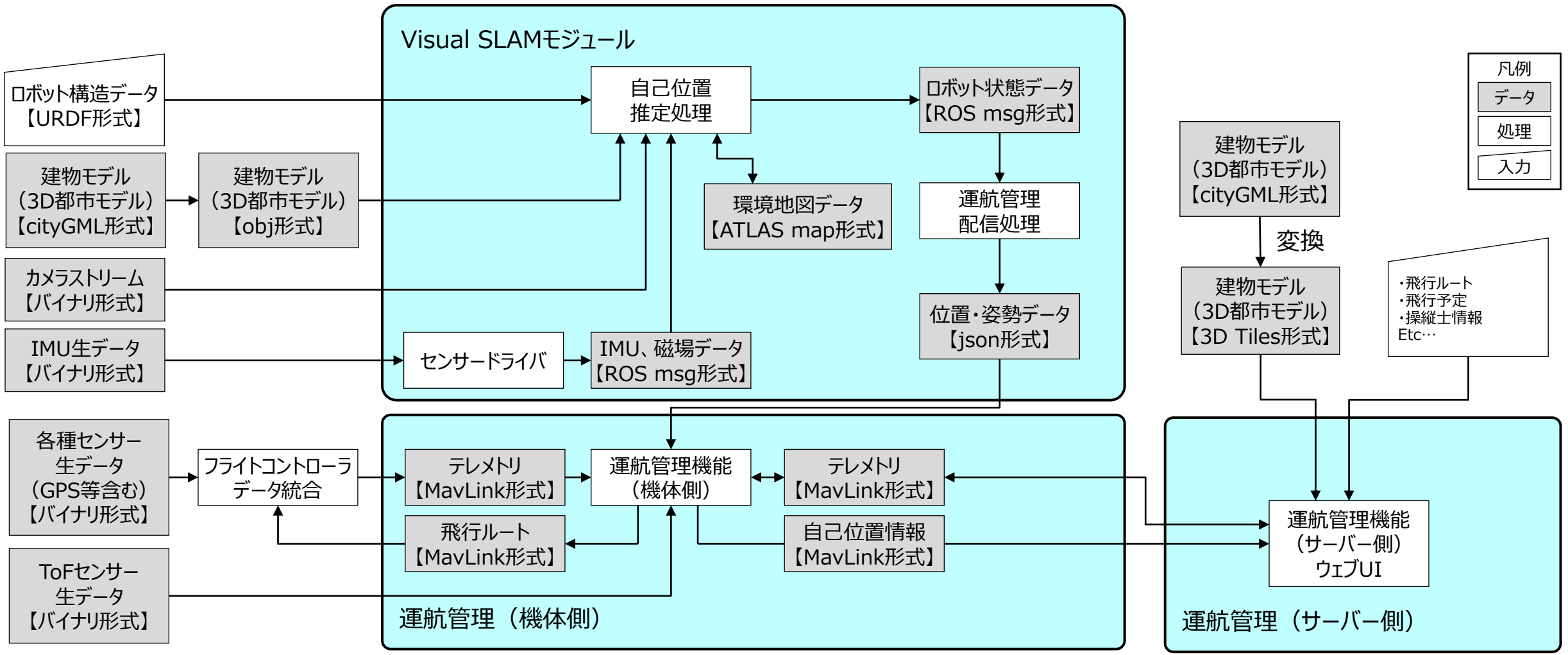
運航管理（サーバー側）のモジュールアーキテクチャ全体図は下記の通り

<凡例> センサー / 既存処理 / 既存ソフト / データ / 開発処理 / 開発ソフト

データアーキテクチャ全体図　LiDAR SLAM



データアーキテクチャ全体図　Visual SLAM



ハードウェアアーキテクチャ

ドローン、コンパニオンコンピュータの仕様は下記の通り

名称	ドローン	コンパニオンコンピュータ
イメージ画像		
メーカー	アトラックラボ社	NVDIA社
モデル・型式	六枚プロペラマルチコプター　HIYOKO18	Jetson Xavier NX
概要・用途	• 産業用の汎用ドローン • 高いカスタマイズ性とメンテナンス性を有するため、本実証で開発するシステムを搭載するベース機体として選定した	• 機体から取得した機体情報（テレメトリ）の運航管理サーバーのデータ転送や、各SLAMによる演算処理を行うための小型コンピュータ • 飛行中にエッジ処理でSLAM演算ができるよう、ドローンに搭載
仕様詳細	• プロペラ径：18インチ • バッテリー：6C（22.2V）、16,000mAh • 最大積載重量：約5kg • 飛行時間：約20分 • フライトコントローラー：Pixhawk	• CPU：6コア NVIDIA Carmel ARM®v8.2 64 ビット CPU 6MB L2+4MB L3 • GPU：48基のTensorコア搭載、384コアNVIDIA Volta™ • メモリ：8GB 128ビット LPDDR4x59.7GB/秒 • ストレージ：16 GB eMMC 5.1

システム要件一覧

システム要件は下記の通り

項目	内容
A1 LiDAR SLAMモジュール	• 3D都市モデルを取り込み、LiDARセンサーをはじめとするドローン搭載のセンサーデータと合わせて自己位置情報を算出する
A2 Visual SLAMモジュール	• 3D都市モデルを取り込み、ステレオカメラをはじめとするドローン搭載のセンサーデータと合わせて自己位置情報を算出する
B 運航管理（機体側）	• LiDAR SLAMやVisual SLAMで算出した自己位置データを運航管理サーバーに転送する • 障害物センサーの情報をもとに障害物を検知し、検知した障害物を自動的に回避する機体制御を行う
C 運航管理（サーバー側）	• 3D都市モデルを組み込み、ユーザー側の3D地図上に機体の位置と3D都市モデルを表示する • 障害物を検知した際、どのような対応するかの選択肢をオペレータに提供する

システム機能一覧 | A1 LiDAR SLAMモジュール

LiDAR SLAMのモジュール詳細は下記の通り

番号	機能名	概要
LS-1	IMUドライバ	IMU測定データ取得用のドライバー
LS-2	IMUフィルタ	IMUが取得したデータに対してフィルタ処理を行う（ノイズ・ゲイン調整等）
LS-3	LiDARドライバ	LiDAR測定データ取得のドライバー
LS-4	LiDARオドメトリ	LiDARからの点群データとIMUデータをもとに自己のオドメトリ情報を算出する
LS-5	自己位置推定（LiDAR）	3次元環境中のロボットの位置を粒子フィルタアルゴリズムで推定する
LS-6	構造データ変換	ロボットの構造データを各種データ統合に使えるように変換する
LS-7	データ統合・変換（Tf2）	各種データの統合・変換処理をしてロボットの状態データを出力する
LS-8	運航管理機能への配信	ROSと運航管理機能の通信およびそれに必要なフォーマット変換

LS-1　IMUドライバ

概要

項目	内容
機能名	• IMUドライバ
概要	• IMU測定データ取得用のドライバー
入力	• IMUセンサーデータ （データストリーム）
出力	• IMUデータ - sensor_msgs, Imu.msg • 磁場データ - sensor_msgs, MagneticField.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイルのイメージ

```
<launch>
   <arg name="output" default="screen"/>
   <arg name="imu" default="imu"/>

   <arg name="port" default="/dev/ttyUSB0" />
   <arg name="frame_id" default="$(arg imu)"/>
   <arg name="baudrate" default="921600"/>
   <arg name="imu_rate" default="100"/>

   <!-- Sync out settings -->
   <!-- to disable this just set it to < 0 -->
   <arg name="sync_rate" default="20"/>
   <arg name="sync_pulse_width_us" default="1000"/>

   <arg name="binary_output" default="true"/>

 <!-- Ros Topic settings -->
   <arg name="enable_mag" default="true"/>
   <arg name="enable_pres" default="true"/>
   <arg name="enable_temp" default="true"/>

   <node pkg="imu_vn_100" name="$(arg imu)" type="imu_vn_100_node" output="$(arg
output)">
      <param name="port" type="string" value="$(arg port)"/>
      <param name="baudrate" type="int" value="$(arg baudrate)"/>

 </node>
</launch>
```

Ⅲ. 実証システム ＞ 4. システム機能
LS-2　IMUフィルタ

概要

項目	内容
機能名	• IMUフィルタ
概要	• IMUが取得したデータに対してフィルタ処理を行う（ノイズ・ゲイン調整等）
入力	• IMUデータ - sensor_msgs, Imu.msg • 磁場データ - sensor_msgs, MagneticField.msg
出力	• IMUデータ（フィルタ処理後） - sensor_msgs, Imu.msg • 磁場データ（フィルタ処理後） - sensor_msgs, MagneticField.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイルのイメージ

```
<launch>
  #### Nodelet manager
############################################################

  <node pkg="nodelet" type="nodelet" name="imu_manager"
    args="manager" output="screen" />

  #### IMU Driver
######################################################################

  <node pkg="nodelet" type="nodelet" name="PhidgetsImuNodelet"
    args="load phidgets_imu/PhidgetsImuNodelet imu_manager"
    output="screen">

    # supported data rates: 4 8 16 24 32 40 ... 1000 (in ms)
    <param name="period" value="4"/>

  </node>

  #### Complementary filter

  <node pkg="imu_complementary_filter" type="complementary_filter_node"
      name="complementary_filter_gain_node" output="screen">
    <param name="do_bias_estimation" value="true"/>
    <param name="do_adaptive_gain" value="true"/>
    <param name="use_mag" value="false"/>
    <param name="gain_acc" value="0.01"/>
    <param name="gain_mag" value="0.01"/>


  </node>
</launch>
```

LS-3　LiDARドライバー

概要

項目	内容
機能名	• LiDARドライバ
概要	• LiDAR測定データ取得のドライバー
入力	• LiDAR取得データパケット
出力	• 点群データ - sensor_msgs, PointCloud2.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイルのイメージ

```
<launch>

  <arg name="ouster_ns" default="ouster" doc="Override the default namespace of all
ouster nodes"/>
  <arg name="sensor_hostname" doc="hostname or IP in dotted decimal form of the
sensor"/>
  <arg name="udp_dest" default=" " doc="hostname or IP where the sensor will send data
packets"/>
  <arg name="lidar_port" default="0" doc="port to which the sensor should send lidar
data"/>
  <arg name="imu_port" default="0" doc="port to which the sensor should send imu
data"/>
  <arg name="udp_profile_lidar" default=" " doc="lidar packet profile; possible values: {
    LEGACY,
    RNG19_RFL8_SIG16_NIR16_DUAL,
    RNG19_RFL8_SIG16_NIR16,
    RNG15_RFL8_NIR8
    }"/>
  <arg name="lidar_mode" default=" " doc="resolution and rate; possible values: {
    512x10,
    512x20,
    1024x10,
    1024x20,
    2048x10,
    4096x5
    }"/>
```

```
</launch>
```

Ⅲ. 実証システム > 4. システム機能
LS-4　LiDARオドメトリ

概要

項目	内容
機能名	• LiDARオドメトリ
概要	• LiDARからの点群データとIMUデータをもとに自己のオドメトリ情報を算出する
入力	• 点群データ - sensor_msgs, PointCloud2.msg • IMUデータ - sensor_msgs, Imu.msg • 磁場データ - sensor_msgs, MagneticField.msg
出力	• 姿勢データ（x,y,z,w） - geometry_msgs, Pose.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整 • Configuration Files（設定ファイル） - トライ＆エラーを行い、適切な自己位置推定をするためのパラメータ調整を実施
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
<launch>
<!-- Launch file for ouster OS2-64 LiDAR -->

    <arg name="rviz" default="true" />

    <rosparam command="load" file="$(find fast_lio)/config/ouster64.yaml" />

    <param name="feature_extract_enable" type="bool" value="0"/>
    <param name="point_filter_num" type="int" value="4"/>
    <param name="max_iteration" type="int" value="3" />
    <param name="filter_size_surf" type="double" value="0.5" />
    <param name="filter_size_map" type="double" value="0.5" />
    <param name="cube_side_length" type="double" value="1000" />
    <param name="runtime_pos_log_enable" type="bool" value="0" />
    <node pkg="fast_lio" type="fastlio_mapping" name="laserMapping"
output="screen" />

    <group if="$(arg rviz)">
    <node launch-prefix="nice" pkg="rviz" type="rviz" name="rviz" args="-d
$(find fast_lio)/rviz_cfg/loam_livox.rviz" />
    </group>
</launch>
```

Configuration Flies (設定ファイル)

```
common:
    lid_topic:  "/os_cloud_node/points"
    imu_topic:  "/os_cloud_node/imu"
    time_sync_en: false         # ONLY turn on when external time synchronization
is really not possible
    ….
```

LS-5 自己位置推定（LiDAR）

概要

項目	内容
機能名	• 自己位置推定（LiDAR）
概要	• 3次元環境中のロボットの位置を粒子フィルタアルゴリズムで推定する
入力	• 点群データ - sensor_msgs, PointCloud2.msg • 姿勢データ（x,y,z,w） - geometry_msgs, Pose.msg • Octomap - octomap_msgs, Octomap
出力	• 位置情報データ（離陸地点からの相対位置） - (geometry_msgs/TransformStamped)
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整 - トライ＆エラーを行い、適切な自己位置推定をするためのや演算粒子数等の最適化を実施（詳細はⅢ-5を参照）
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
<?xml version="1.0"?>
<launch>

  <arg name="map_name_path"/>
  <arg name="init_x"/>
  <arg name="init_y"/>
  <arg name="init_z"/>
  <arg name="init_a"/>
  <arg name="num_particles"/>
  <arg name="alpha"/>
  <arg name="take_off_height"/>

  <node pkg="amcl3d" type="amcl3d_node" name="amcl3d_node"
output="screen">

    <!-- Topics and frames -->
    <!-- To set the topic which has information about the point cloud from the
robot view -->
    <remap from="laser_sensor" to="/lidar_pointcloud"/>

    <!-- To set the topic which has information about the robot odometry -->
    <remap from="odometry" to="/vicon_odometry"/>

    <!-- To set the topic which has information about the sensor-range sensors --
>
    <remap from="radiorange_sensor" to="/nanotron_node/range"/>

  </node>
</launch>.
```

LS-6　構造データ変換

概要

項目	内容
機能名	• 構造データ変換
概要	• ロボットの構造データを各種データ統合に使えるように変換する
入力	• ロボット構造データ - urdf file
出力	• ロボットの状態データ(std_msgs/msg/String) • 変換用データ (tf2_msgs)
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
import os
import subprocess

import launch
import launch_ros.actions
from launch_ros.substitutions import FindPackageShare


def generate_launch_description():
    pkg_share =
FindPackageShare('robot_state_publisher').find('robot_state_publisher')
    urdf_dir = os.path.join(pkg_share, 'urdf')
    xacro_file = os.path.join(urdf_dir, 'test-desc.urdf.xacro')
    p = subprocess.Popen(['xacro', xacro_file], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    robot_desc, stderr = p.communicate()
    params = {'robot_description': robot_desc.decode('utf-8')}
    rsp = launch_ros.actions.Node(package='robot_state_publisher',
                                  executable='robot_state_publisher',
                                  output='both',
                                  parameters=[params])

    return launch.LaunchDescription([rsp])
```

LS-7　データ統合・変換

概要

項目	内容
機能名	• データ統合・変換
概要	• 各種データの統合・変換処理をしてロボットの状態データを出力する
入力	• ロボットの状態データ - std_msgs/msg/string • 変換用データ - tf2_msgs • 位置情報データ - geometry_msgs - TransformStamped
出力	• ロボットの状態データ （std_msgs/msg/String）
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
<launch>
  <test test-name="transform_listener_time_reset_test" pkg="tf2_ros"
type="tf2_ros_test_time_reset" />
  <param name="/use_sim_time" value="True"/>
</launch>
```

LS-8　運行管理機能への配信

概要

項目	内容
機能名	• データ統合・変換
概要	• ROSと運航管理機能の通信およびそれに必要なフォーマット変換
入力	• ロボットの状態データ - std_msgs/msg/String
出力	• 自己位置情報データ - json • 自己姿勢データ - json
新規開発要素	• スクラッチから作成した新規ROSノード • 各種データ処理、および運航管理機能（機体側）との通信に必要なコードを作成
プラットフォーム	• ROS

ROSパッケージのデータ構造

```
- src
      - ROS_node
            |- msg
            |          - message.msg
            |- sctipts
            |          - cosmos.py
            |- package.xml
            |- CMakeLists.txt
```

massage.msg: ノード間でやり取りされるメッセージの型定義

cosmos.py: データ処理のメインコード（スクリプト）
自己位置推定結果をROSのmessageとして受け取り、運航管理機能（機体側）で利用できるようjsonファイルとしてコンバートするスクリプト

package.xml: パッケージの基本情報やROSノードとして立ち上げる際に必要になる依存パッケージ等を記載したファイル

CMakeLists.txt: ROSノードとしてビルドする際の依存関係や設定値を記述したファイル

システム機能一覧 | A2 Visual SLAMモジュール

LiDAR SLAMのモジュール詳細は下記の通り

番号	機能名	概要
VS-1	IMUドライバ	IMU測定データ取得用のドライバー
VS-2	IMUフィルタ	IMUが取得したデータに対してフィルタ処理を行う（ノイズ・ゲイン調整等）
VS-3	シミュレータ	CityGMLを用いてVisual SLAMに必要な事前環境地図を作製するための仮想カメラ出力の作成を行う
VS-4	自己位置推定（Visual）	Orb SLAMアルゴリズムを用いたカメラの画像データからの地図および自己位置推定
VS-5	構造データ変換	ロボットの構造データを各種データ統合に使えるように変換する
VS-6	データ統合・変換（Tf2）	各種データの統合・変換処理をしてロボットの状態データを出力する
VS-7	運航管理機能への配信	ROSと運航管理機能の通信およびそれに必要なフォーマット変換

VS-1 IMUドライバ

概要

項目	内容
機能名	• IMUドライバ
概要	• IMU測定データ取得用のドライバー
入力	• IMUセンサーデータ （データストリーム）
出力	• IMUデータ - sensor_msgs, Imu.msg • 磁場データ - sensor_msgs, MagneticField.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイルのイメージ

```
<launch>
   <arg name="output" default="screen"/>
   <arg name="imu" default="imu"/>

   <arg name="port" default="/dev/ttyUSB0" />
   <arg name="frame_id" default="$(arg imu)"/>
   <arg name="baudrate" default="921600"/>
   <arg name="imu_rate" default="100"/>

   <!-- Sync out settings -->
   <!-- to disable this just set it to < 0 -->
   <arg name="sync_rate" default="20"/>
   <arg name="sync_pulse_width_us" default="1000"/>

   <arg name="binary_output" default="true"/>

 <!-- Ros Topic settings -->
   <arg name="enable_mag" default="true"/>
   <arg name="enable_pres" default="true"/>
   <arg name="enable_temp" default="true"/>

   <node pkg="imu_vn_100" name="$(arg imu)" type="imu_vn_100_node" output="$(arg
output)">
      <param name="port" type="string" value="$(arg port)"/>
      <param name="baudrate" type="int" value="$(arg baudrate)"/>

 </node>
</launch>
```

VS-2 IMUフィルタ

概要

項目	内容
機能名	• IMUフィルタ
概要	• IMUが取得したデータに対してフィルタ処理を行う（ノイズ・ゲイン調整等）
入力	• IMUデータ - sensor_msgs, Imu.msg • 磁場データ - sensor_msgs, MagneticField.msg
出力	• IMUデータ（フィルタ処理後） - sensor_msgs, Imu.msg • 磁場データ（フィルタ処理後） - sensor_msgs, MagneticField.msg
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイルのイメージ

```
<launch>
  #### Nodelet manager
############################################################

  <node pkg="nodelet" type="nodelet" name="imu_manager"
    args="manager" output="screen" />

  #### IMU Driver
######################################################################

  <node pkg="nodelet" type="nodelet" name="PhidgetsImuNodelet"
    args="load phidgets_imu/PhidgetsImuNodelet imu_manager"
    output="screen">

    # supported data rates: 4 8 16 24 32 40 ... 1000 (in ms)
    <param name="period" value="4"/>

  </node>

  #### Complementary filter

  <node pkg="imu_complementary_filter" type="complementary_filter_node"
      name="complementary_filter_gain_node" output="screen">
    <param name="do_bias_estimation" value="true"/>
    <param name="do_adaptive_gain" value="true"/>
    <param name="use_mag" value="false"/>
    <param name="gain_acc" value="0.01"/>
    <param name="gain_mag" value="0.01"/>


  </node>
</launch>
```

VS-3　シミュレータ

概要

項目	内容
機能名	• シミュレータ
概要	• CityGMLを用いてVisual SLAMに必要な事前環境地図を作製するための仮想カメラ出力の作成を行う
入力	• 3D都市モデル - obj file
出力	• 仮想ステレオカメラ取得データパケット • IMUデータ - sensor_msgs - Imu.msg
新規開発要素	• Unityに3D都市モデルを取り込み、指定したルートを飛行した場合の仮想カメラの映像や軌跡データを出力する環境を構築
プラットフォーム	• Unity

シミュレータイメージ

Unity上にctyGMLを配置（仮想カメラのルート設定）

仮想カメラの映像

VS-4 自己位置推定（Visual）

概要

項目	内容
機能名	• 自己位置推定（Visual）
概要	• Orb SLAMアルゴリズムを用いたカメラの画像データからの地図および自己位置推定
入力	• ステレオカメラ取得データパケット • IMUデータ - sensor_msgs, Imu.msg • 地図データ - ATLAS map
出力	• 地図データ - ATLAS map • 位置情報データ （離陸地点からの相対位置） - geometry_msgs, TransformStamped
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整 • Configuration Files（設定ファイル） - トライ＆エラーを行い、適切な自己位置推定をするためのパラメータ調整を実施
プラットフォーム	• ROS

シミュレータイメージ

Launch Files (構成ファイル)

```
<launch>
   <!-- ORB-SLAM3 -->
   <node name="orb_slam3_mono_inertial" pkg="orb_slam3_ros_wrapper"
type="orb_slam3_ros_wrapper_mono_inertial" output="screen">
      <!-- From realsense2_camera node -->
      <remap from="/camera/image_raw"
to="/camera/fisheye1/image_raw"/>
      <remap from="/imu"                    to="/camera/imu"/>

      <!-- Parameters for original ORB-SLAM3 -->
      <param name="voc_file"      type="string"   value="$(find
orb_slam3_ros_wrapper)/config/ORBvoc.txt" />
      <param name="settings_file" type="string"   value="$(find
orb_slam3_ros_wrapper)/config/Realsense_T265.yaml" />


   </node>
</launch>
```

Configuration Flies (設定ファイル)

```
File.version: "1.0"

Camera.type: "PinHole"

# Right Camera calibration and distortion parameters (OpenCV)
Camera1.fx: 617.201
Camera1.fy: 617.362
Camera1.cx: 324.637
Camera1.cy: 242.462
…
```

VS-5　構造データ変換

概要

項目	内容
機能名	• 構造データ変換
概要	• ロボットの構造データを各種データ統合に使えるように変換する
入力	• ロボット構造データ - urdf file
出力	• ロボットの状態データ(std_msgs/msg/String) • 変換用データ (tf2_msgs)
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
import os
import subprocess

import launch
import launch_ros.actions
from launch_ros.substitutions import FindPackageShare


def generate_launch_description():
    pkg_share =
FindPackageShare('robot_state_publisher').find('robot_state_publisher')
    urdf_dir = os.path.join(pkg_share, 'urdf')
    xacro_file = os.path.join(urdf_dir, 'test-desc.urdf.xacro')
    p = subprocess.Popen(['xacro', xacro_file], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    robot_desc, stderr = p.communicate()
    params = {'robot_description': robot_desc.decode('utf-8')}
    rsp = launch_ros.actions.Node(package='robot_state_publisher',
                                  executable='robot_state_publisher',
                                  output='both',
                                  parameters=[params])

    return launch.LaunchDescription([rsp])
```

VS-6　データ統合・変換

概要

項目	内容
機能名	• データ統合・変換
概要	• 各種データの統合・変換処理をしてロボットの状態データを出力する
入力	• ロボットの状態データ - std_msgs/msg/string • 変換用データ - tf2_msgs • 位置情報データ - geometry_msgs - TransformStamped
出力	• ロボットの状態データ （std_msgs/msg/String）
新規開発要素	• Launch Files（構成ファイル） - ROSメッセージの配信設定およびデータレート等を調整
プラットフォーム	• ROS

構成ファイル、設定ファイルのイメージ

Launch Files (構成ファイル)

```
<launch>
  <test test-name="transform_listener_time_reset_test" pkg="tf2_ros"
type="tf2_ros_test_time_reset" />
  <param name="/use_sim_time" value="True"/>
</launch>
```

VS-7　運行管理機能への配信

概要

項目	内容
機能名	• 運行管理機能への配信
概要	• ROSと運航管理機能の通信およびそれに必要なフォーマット変換
入力	• ロボットの状態データ - std_msgs/msg/String
出力	• 自己位置情報データ - json • 自己姿勢データ - json
新規開発要素	• スクラッチから作成した新規ROSノード • 各種データ処理、および運航管理機能（機体側）との通信に必要なコードを作成
プラットフォーム	• ROS

ROSパッケージのデータ構造

```
- src
      - ROS_node
            |- msg
            |          - message.msg
            |- sctipts
            |          - cosmos.py
            |- package.xml
            |- CMakeLists.txt
```

massage.msg: ノード間でやり取りされるメッセージの型定義

cosmos.py: データ処理のメインコード（スクリプト）
自己位置推定結果をROSのmessageとして受け取り、運航管理機能（機体側）で利用できるようjsonファイルとしてコンバートするスクリプト

package.xml: パッケージの基本情報やROSノードとして立ち上げる際に必要になる依存パッケージ等を記載したファイル

CMakeLists.txt: ROSノードとしてビルドする際の依存関係や設定値を記述したファイル

システム機能一覧 | B 運航管理：機体側

運航管理（機体側）モジュール機能は下記の通り

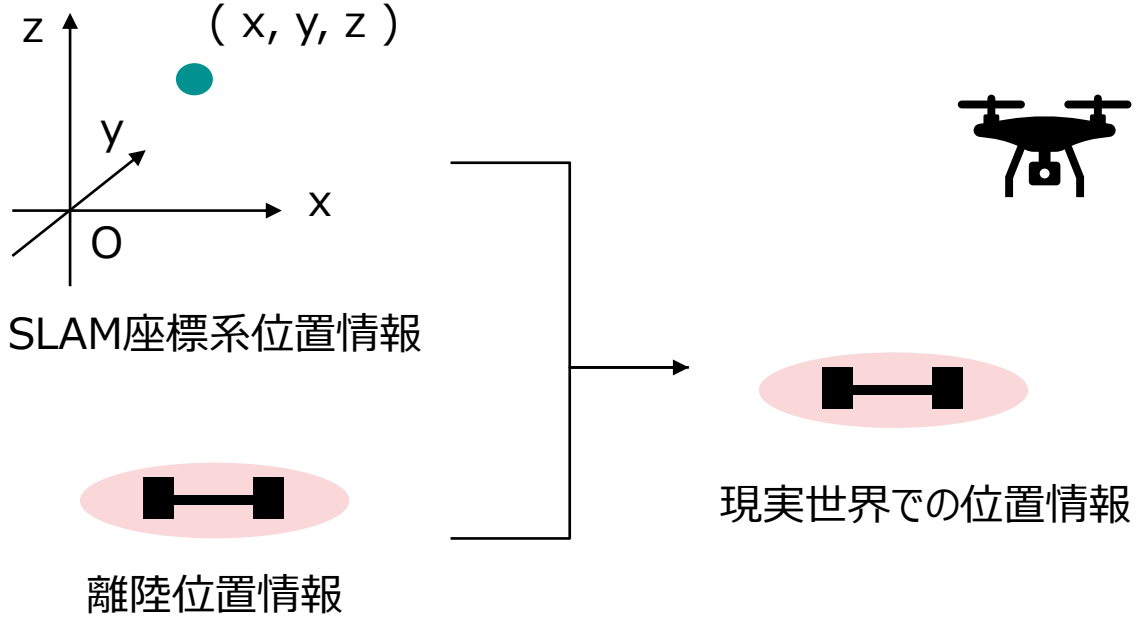
番号	機能名	概要
B-1	SLAMインターフェース	SLAMと運航管理機能のデータの受渡し・データ変換（移動データ→3D地図上の航路データ）等を行う
B-2	テレメトリ	IMUが取得したデータに対してフィルタ処理を行う（ノイズ・ゲイン調整等）
B-3	航路計画	距離センサーの情報と機体の進行方向をもとに、一時停止指令、障害物回避ルートの生成を行う
B-4	距離センサーデータ処理	距離センサー取得情報を統合し、メートル単位に変換したうえで進行方向に対する障害物の配置を演算する
B-5	ミッション調整	障害物が検出された際に、フライトプランを調整し、フライトコントローラーへの転送を行う

B-1　SLAMインターフェース

概要

項目	内容
機能名	• SLAMインターフェース
概要	• SLAMと運航管理機能のデータの受渡し・データ変換（移動データ→3D地図上の航路データ）等を行う
入力	• SLAM位置情報 • 現在位置情報 • ホームポイントの位置情報 • 機体の進行方向
出力	• 位置情報データ (Dictionary)
新規開発要素	• 運航管理（機体側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

SLAMインターフェスの機能イメージ



- SLAMモジュールからの出力はjsonファイルとして運航管理機能（機体側）に受け渡される
- SLAMモジュールの演算は離陸位置を基準としたx,y,zの直交座標系でなされる
- SLAMインターフェーズモジュールでは、離陸点位置情報とSLAM出力を統合して、現実世界での位置情報に変換する

Ⅲ. 実証システム ＞ 4. システム機能
B-2　テレメトリ

概要

項目	内容
機能名	• テレメトリ
概要	• 機体やセンサー情報の集積や必要なコマンドのハンドリング
入力	• 機体情報、センサー入力、運航管理サーバーからのコマンドやミッション情報
出力	• テレメトリ情報 (Json)
新規開発要素	• 既存のテレメトリ処理機能に対し、SLAMからの位置情報をパラメータとして追加する改修を実施
プラットフォーム	• COSMOS

テレメトリ情報の一例（飛行ログより抽出）

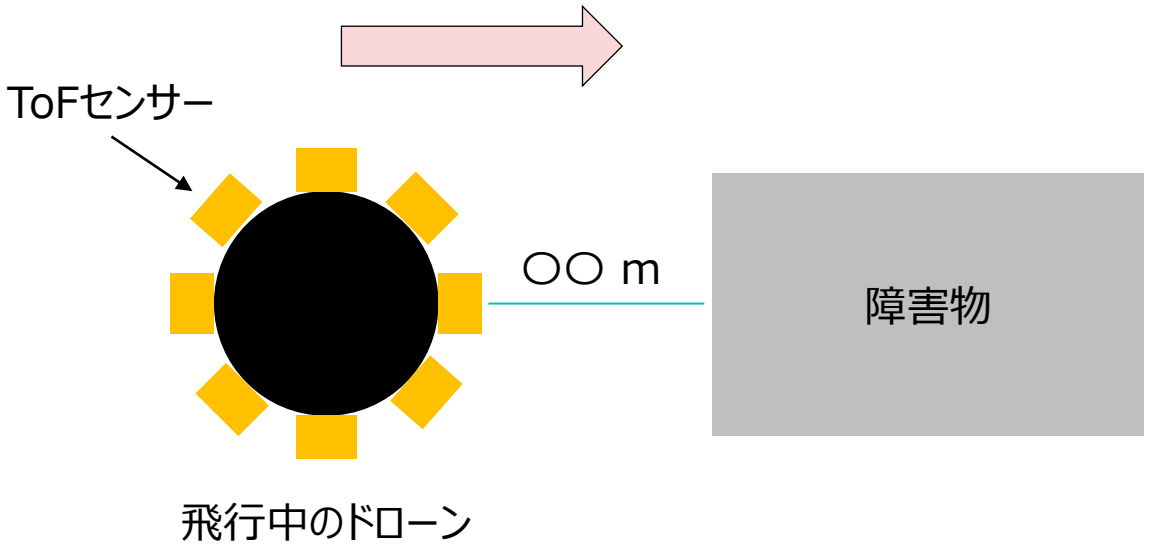

```
mode: LOITER
armed: false
gps_0:
    fix: 6
    HDOP: 75
    VDOP: 117
    numsat: 15
stamp: 1677568034127722
state: STANDBY
ekf_ok: true
gimbal:
    yaw: 0
    roll: 0
    pitch: 0
battery:
    level: 99
    current: 0
    voltage: 22.238
heading: 168
mission:
    id: 7896
activity: Mission Ongoing (WP 11/17)
attitude:
    yaw: 2.9324986934661865
    roll: 0.00228208489716053
    pitch: 0.00045481626875698566
location:
    home:
```

B-3　航路計画

概要

項目	内容
機能名	• 航路計画
概要	• 距離センサーの情報と機体の進行方向をもとに、一時停止指令を行う
入力	• 距離センサー情報 • 機体進行方向
出力	• 障害物情報 • 一時停止信号
新規開発要素	• 運航管理（機体側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

航路計画機能イメージ



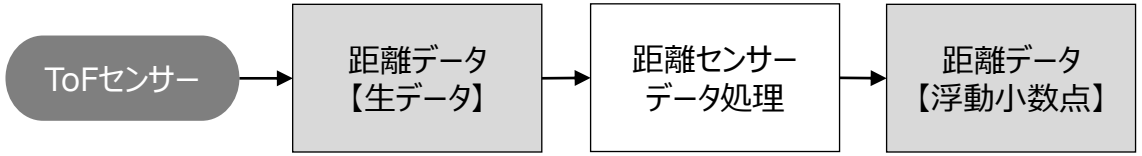
- ToFセンサーから得られた情報および機体の進行方向情報の元に進行方向の障害物を検出する
- 進行方向の規定値以内の距離に障害物が検出された場合、機体を一時停止させる信号を発令

B-4　距離センサーデータ処理

概要

項目	内容
機能名	• 距離センサーデータ処理
概要	• 距離センサー取得情報を統合し、メートル単位に変換したうえで進行方向に対する障害物の配置を演算する
入力	• 距離センサー計測生データ（シリアル）
出力	• 距離センサーデータ
新規開発要素	• 運航管理（機体側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

航路計画機能イメージ



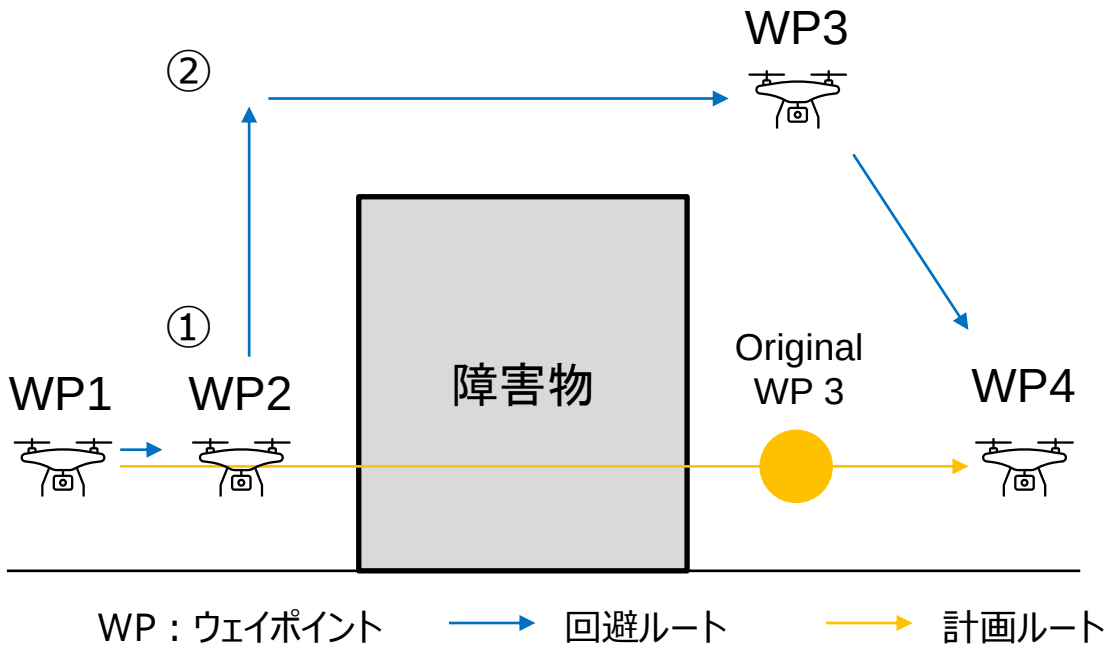
- シリアルデータとして取得されたToFセンサーの生データをメートル単位の計測データに変換する。
- 機体に取り付けられた8つのToFセンサーとその取り付け位置から機体の機首方向に対してどの方向にどれくらいの距離の障害物があるのかを演算する。

B-5　ミッション調整

概要

項目	内容
機能名	• ミッション調整
概要	• 障害物が検出された際に、フライトプランを調整し、フライトコントローラーへの転送を行う
入力	• 飛行ルート （運航管理機能（サーバー側）の出力）
出力	• 飛行指示 （フライトコントローラへの出力）
新規開発要素	• 運航管理（機体側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

ミッション調整イメージ



① 障害物を検知し、一時停止する（B-3の機能）
　→ 運航管理機能（サーバー側）から自動回避指示を受けたら上昇を行う

② 障害物が検知できなくなるまで上昇。十分な高さマージンが確保出来たら次のウェイポイントを目指す

システム機能一覧 | C 運航管理：サーバー側

運航管理（機体側）モジュール機能は下記の通り

番号	機能名	概要
C-1	3D地図表示	3D地図を飛行監視画面面に表示する
C-2	NDEによる機体トラック	NDE座標系を用いて、自己位置を演算した上で、飛行ルートをNDE座標系でトラックする
C-3	障害物検知	障害物を検知した際に、オペレーターに停止、手動での操作、自動での回避ルート作成の選択肢を表示する

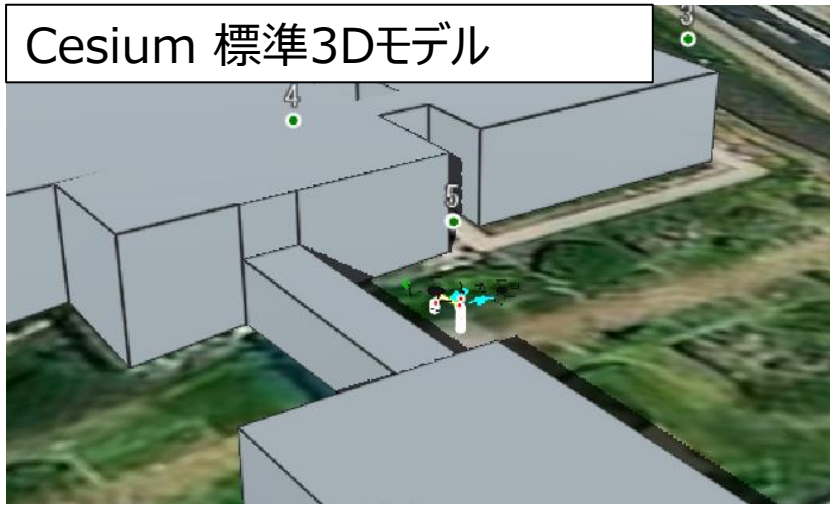
Ⅲ. 実証システム ＞ 4. システム機能
C-1　3D地図表示

概要

項目	内容
機能名	• 3D地図表示
概要	• 3D地図を飛行監視画面に表示する
入力	• 3D都市モデル（3D Tile） • Cesium標準3Dモデル
出力	• なし （ユーザーインターフェースでの画面表示）
新規開発要素	• 表示する3DモデルをCesium標準モデルに加えて3D都市モデルを選択表示できる機能を追加
プラットフォーム	• COSMOS

Cesium標準と3D都市モデルの3D都市モデル比較

Cesium 標準3Dモデル



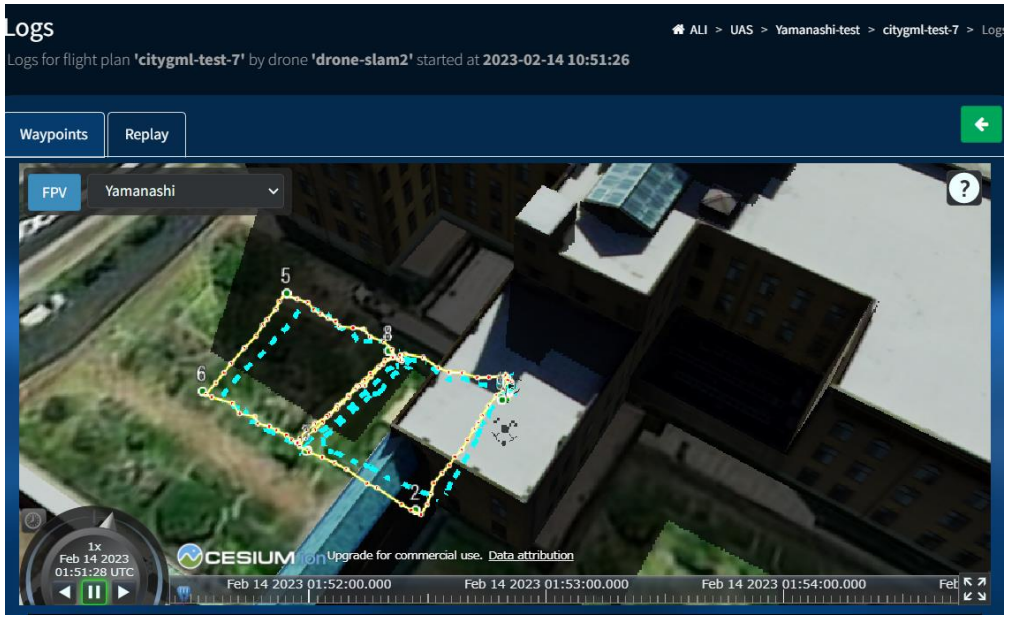
3D都市モデル（LOD2）

C-2　NEDによる機体トラック

概要

項目	内容
機能名	• NEDによる機体トラック
概要	• NED座標系を用いて、自己位置を演算した上で、飛行ルートをNDE座標系でトラックする
入力	• ホームポジション • 機体情報（テレメトリ）
出力	• 地図上におけるフライトパス
新規開発要素	• 運航管理（サーバー側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

Cesium標準と3D都市モデルの3D都市モデル比較



- COSMOSにおいて、GNSS（RTK含む）による推定飛行ルートの軌跡に加え（上図黄色線）に加え、SLAMによる推定飛行ルートの軌跡を表示する機能を実装（上図水色点線）
- SLAMの自己位置推定は直交座標系においてなされるため、GNSSの表示に用いる緯度経度によるトラッキング機能だけでなく、直交座標系であるNDE（North-East-Down）座標系による機体位置トラックの機能を実装

C-3　障害物検知

概要

項目	内容
機能名	• 障害物検知
概要	• 障害物を検知した際に、オペレーターに停止、手動での操作、自動での回避ルート作成の選択肢を表示する
入力	• ウェイポイント情報 • 機体位置 • 障害物位置
出力	• 画面表示（UI） • 回避ルート （運航管理機能（機体側）への出力）
新規開発要素	• 運航管理（サーバー側）の追加機能としてコードを新規作成・実装
プラットフォーム	• COSMOS

障害物回避選択UI

- 運航管理機能（機体側）より障害物検知の情報を受領した場合、UIにてオペレータにアクション選択を表示
  - 機体は運航管理機能（機体側）により一時停止している
- 自動回避を選択した場合、回避ルート情報を運航管理機能（機体側）に出力として送信する
（障害物回避の考え方はⅢ-4 B-5を参照）

アルゴリズム一覧

名称	説明
LiDAR SLAM	• ドローンに取り付けられたLiDARセンサーによって取得した周辺環境の情報を元に自己位置推定と周辺地図生成を同時に行う。 • LiDAR点群情報や各種センサー（IMU等）の情報を元にオドメトリを算出し、そのオドメトリとLiDAR点群を組み合わせることで自己位置推定を行う。 • 自己位置推定を行う際、位置推定精度向上のため、取得した点群データと事前にインプットした周辺環境の地図情報を照合する。
Visual SLAM	• ドローンに取り付けられたステレオカメラによって取得した画像データをもとに、自己位置推定と周辺地図生成を同時に行う。 • カメラによって取得した画像データから特徴点（角やエッジなどの特徴的な点）を抽出し、その特徴点の推移とIMUなどの情報をもとに自己の位置・姿勢情報を推定する。 • 特徴点情報はSLAMのシステム内に保存され、地図の統合やループとじ込みなどの処理に使われる。

LiDAR SLAMによる自己位置推定のフロー

LiDAR SLAMを用いた自己位置推定の作業フローは下記の通り

① 事前地図の作成

- 3D都市モデルをLiDAR SLAMの事前地図として活用するために、BlenderやROSのPCLおよびOctomapライブラリを利用してobj形式→Octomap形式に変換する

② LiDARオドメトリの算出

- 自己位置推定の演算のファーストステップとして、LiDARセンサーを搭載したドローンの移動量や移動変位（LiDARオドメトリ）を、「FAST LIO2」というライブラリを用いて算出する
- IMU情報とLiDARセンサーで取得した点群データをもとに、周辺環境の特徴点分布から移動経路を演算する

③ 事前地図との整合による自己位置推定

- 算出したオドメトリ情報、事前にインストール・変換した3D都市モデル（Octomap形式）、LiDARセンサーで取得した点群データをもとに自己位置推定を行う
- 自己位置推定の方法として、 粒子フィルタと呼ばれる手法を採用（AMCL）

利用したライブラリ一覧

名称	説明
FAST LIO2	• LiDAR点群情報とセンサー情報（IMU）をもとにオドメトリ算出を行うアルゴリズム • LiDAR点群データより抽出した特徴点情報とIMUから取得される加速度・ジャイロの情報をベースに、主にカルマンフィルタを用いて自己の位置・姿勢の変化量を推定するアルゴリズム
AMCL （Adaptive Monte Carlo Localization）	• オドメトリ情報とLiDARセンサー、事前地図の情報を元に自己位置の推定を行うLiDAR SLAMのアルゴリズム • 領域上に多数の粒子を仮定し、取得した点群データと事前地図を比較して各粒子の尤度を算出し、粒子の重みづけ平均として自己位置の推定を行う

Ⅲ. 実証システム ＞ 5. アルゴリズム ＞ LiDAR SLAM
① 事前地図の作成（3D都市モデルの活用により効率化可能な業務）

SLAMにおいては、事前に環境地図を与えることで自己位置推定の精度や安定性が向上する。本実証では事前地図構築への3D都市モデル活用や必要性をを以下に示す（3D都市モデル活用詳細はⅢ-6-② Octomap化のデータ処理フローを参照）

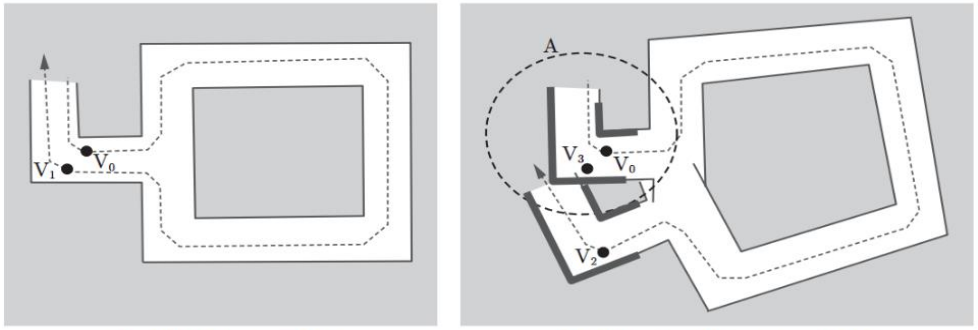
事前地図とは

SLAMにおける課題

- 各種センサーの誤差や周辺環境の時間的な変化、推定誤差などの様々な要因により推定自己位置と周辺環境地図の両方にある程度の誤差が発生
  - 例）部屋を一周して同じ位置に戻ってきたロボットが、誤差のために初期位置と同じ場所にいるということを認識できない（下図*）

事前地図による対策アプローチ

- 自己位置と周辺環境の両方が未知であるSLAM処理のうち、ある程度確からしい事前地図（周辺環境情報）を用いることで、周辺環境マッピングの精度を上げ、全体としての精度の向上を目指す



(a) 実世界でのループ経路　(b) 地図での再訪点のマッチング

Source: 友納　正裕「SLAM入門 - ロボットの自己位置推定と地図構築の技術」 オーム社（2018年）

事前地図の利用方法

事前地図の利用方法

- 静的活用
  - 与えられた事前地図を真としてその事前地図を常に利用する
- 動的活用
  - 与えられた事前地図を参照しつつも実際にセンサーから得られた情報をもとに逐次地図を更新

本実証のLiDAR SLAMで利用する静的活用方法の特徴

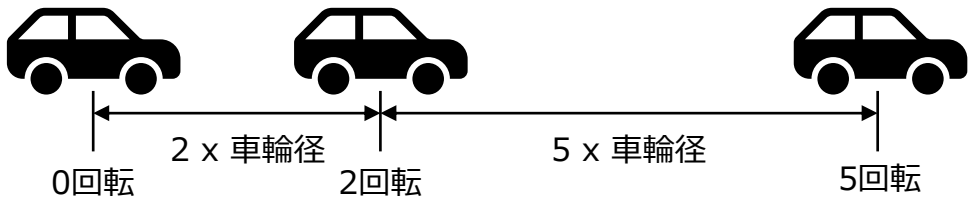
- 十分に精度が高く（現実に近く）安定した事物の事前地図があれば安定した結果が期待できる
  - 一時的に配置されている事物（通行人や車両等）などはSlamの運用時にセンサーのデータとして検出されるが、それらが事前地図に反映されることは繰り返し活用される環境データとしては望ましくない（自己位置推定のノイズとなってしまうため）
  - このように、随時更新を行わずに事前地図を活用することは自己位置推定の安定に寄与すると考えられる
- 事物の経年変化に対応できない
  - 自己位置推定を行う環境自体が永続的に変化した場合（例えば建物がなくなったり、災害等による地形変化など）、それを反映することができないため、自己位置推定にネガティブな影響を与えてしまう可能性がある

② LiDARオドメトリの算出

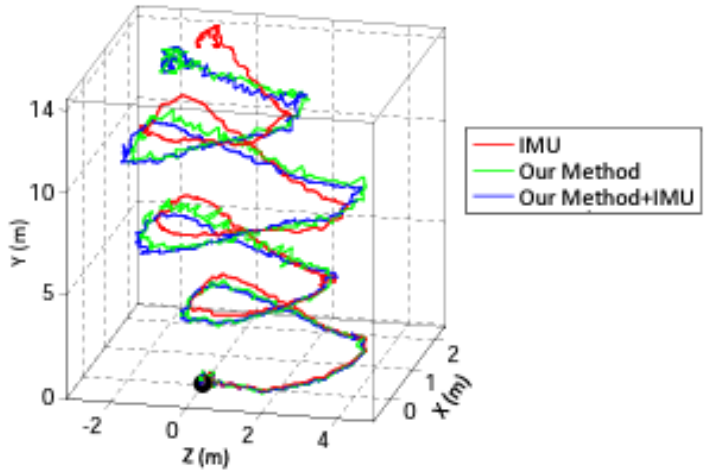
IMU情報とLiDARによる点群データをもとにオドメトリを算出し、周辺環境の特徴点分布から移動経路を演算する

オドメトリとは

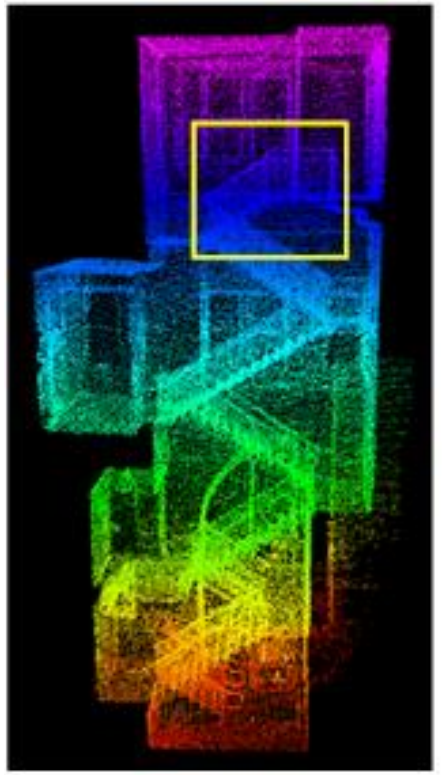
- オドメトリとは、モーションセンサー等を用いてロボットの位置や姿勢の変化を時系列で推定すること、またはそのようにして得られた位置・姿勢の変化量情報である
- 例：車輪により走行するロボットの位置変化を、車輪に取り付けたロータリエンコーダの情報をもとに推定する
- 速度や加速度の時系列情報の積み重ねによる推定を行うため、センサー誤差やサンプリング周期などの影響を強く受けるという特性を持つ
- ロボットに取り付けたカメラの映像やLiDARセンサーを用いて得られた特徴点の移動量からオドメトリを算出するビジュアルオドメトリやLiDARオドメトリなどの手法もある



算出結果のイメージ*



↑　LiDARオドメトリの算出結果
（IMU単独、LiDAR+IMUの結果を併記）



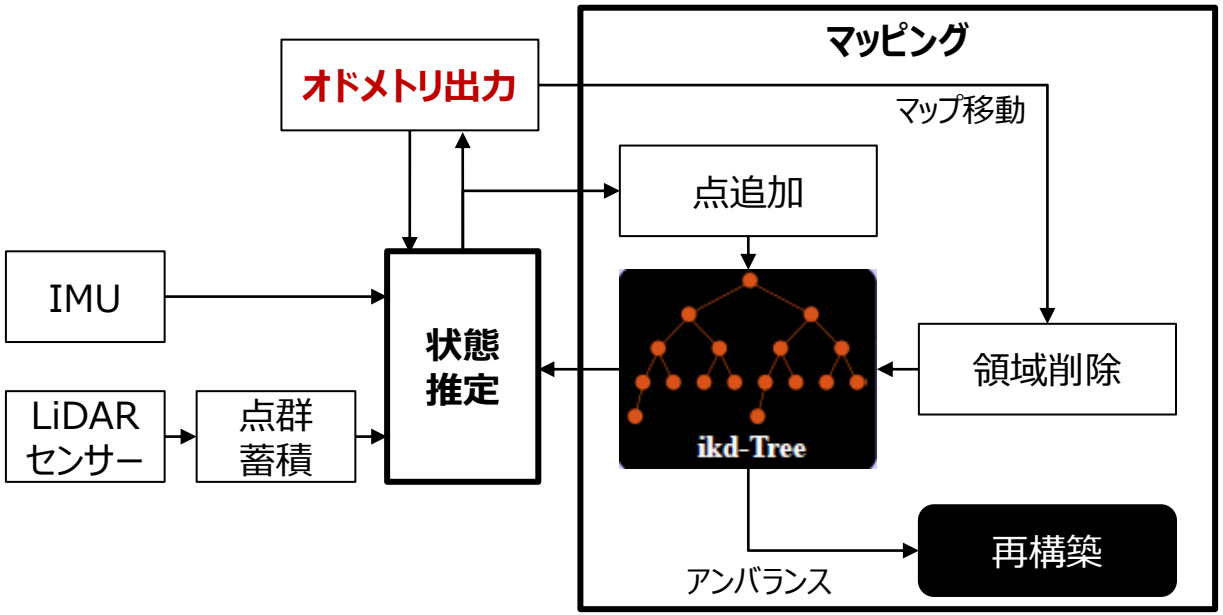
→
オドメトリを算出した際の周辺状況

*: Zhang, J. & Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. In Robotics: Science and Systems Conference (RSS), Berkeley, CA.

FAST LIO2について

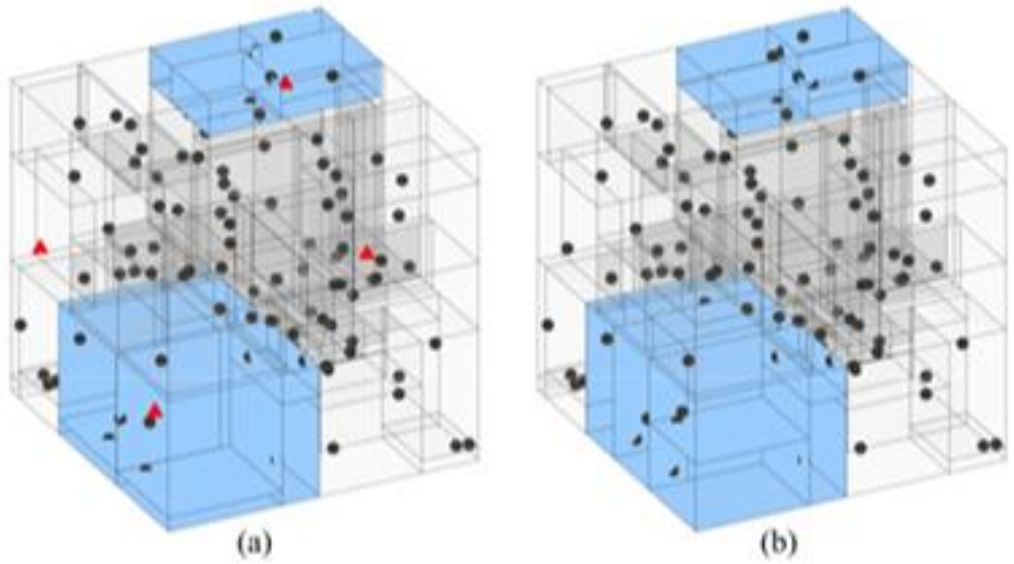
FAST LIO2では、データの追加・削除等の操作が可能なikd-Tree構造を点群マッピングに用いることで、効率的に演算処理することができる

FAST LIO2（LiDAR LIO2）のシステム構成*1



- LiDAR LIO2とは、LiDARオドメトリの算出に用いるアルゴリズムである
- IMUによって取得したジャイロ・加速度情報とLiDARセンサーによって取得した点群データから、主にカルマンフィルタを活用して自己位置の移動量を演算し、アルゴリズム内部で保持している点群データ情報との比較を行う

ikd Treeにおけるデータ追加とリバランスの模式図*2



- ikd Tree（incremental k dimension Tree）とは、k次元のデータを分類する空間分割データ構造であるkd-Treeに対して、データの追加や削除などの動的な操作を可能にしたものである
- FAST LIO2においては、取得した点群データとアルゴリズム内部で保持している点群データとの照合に用いる

*1: Wei Xu et.al. FAST-LIO2: Fast Direct LiDAR-Inertial Odometry (2022) IEEE Transaction on Robotics, Volume 38 Issue:4 をもとに作成
*2: Yixi Cai et.al ikd-Tree: An Incremental K-D Tree for Robotic Applications (2021) eprint arXiv:2102.10808

③ 事前地図との整合による自己位置推定

LiDAR SLAMにおける自己位置推定のアルゴリズムは手法により詳細はさまざまに異なるが、下記に基本的な考え方を記する。

地図と自己位置推定の算出プロセス

LiDAR点群データの取得

- LiDARセンサーを用いてLiDAR位置を原点とする周辺点群データを取得する

センサー姿勢データの取得

- IMUなどをもとに、センサーの姿勢データを取得する。
- オドメトリを活用して姿勢だけでなく位置情報推定まで行う場合もある

点群データの座標変換

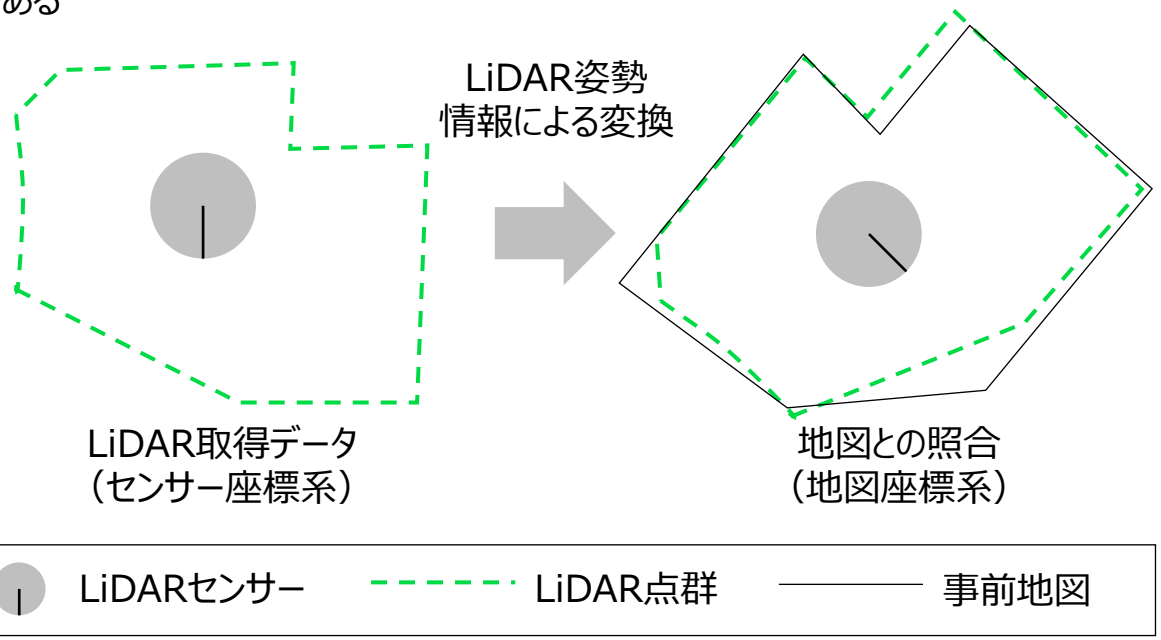
- LiDAR姿勢データをもとに、取得したLiDAR点群データを地図側の座標系に変換する

事前地図との照合

- 地図座標系に変換された点群データと事前地図を比較することで自己位置を推定する。

地図との照合

- 事前地図との照合は、LiDAR点群によって得られた周辺環境と事前地図とを比較し、最小二乗法などの方法でその誤差が最小になる位置を自己位置推定値として用いる
- 各点群に対して誤差を算出する場合もあるが、点群から角やエッジなどの特徴点を抽出して（ランドマークと呼ばれる）その特徴点を用いて評価する場合もある

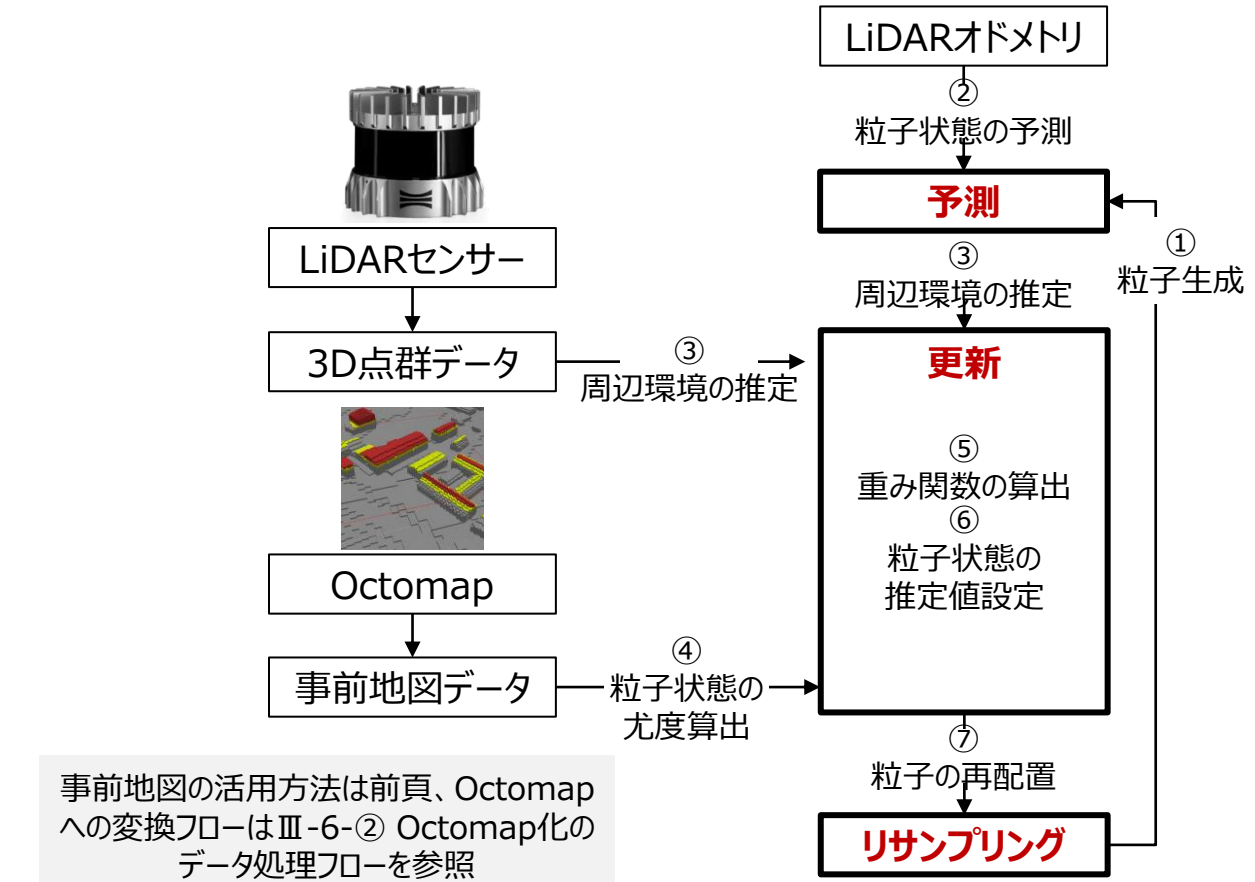


AMCL（Adaptive Monte Carlo Localization）について

自己位置推定精度を高くするために、自己位置や姿勢を多数の粒子とその尤度（重み関数）で推定するAMCLというアルゴリズムを採用

①粒子生成	• 空間上に多数の粒子を生成する • 各粒子の状態（位置・姿勢）を定義する
②粒子状態の予測	• LiDARオドメトリの情報をもとに、次のステップにおける各粒子の状態を予測する
③周辺環境の推定	• 各粒子において予測された状態情報と、LiDARセンサーから取得したデータをもとに、周辺環境を推定する
④粒子状態の尤度算出	• Octomapからの事前地図データと周辺環境推定の結果とを比較して、各粒子の状態の尤度を算出する
⑤重み関数の算出	• 全粒子の尤度をもとに、各粒子の尤度を正規化した重み関数を算出する
⑥粒子状態の推定値設定	• 全粒子の重み付け平均を現粒子状態（位置・姿勢）の推定値として設定する
⑦粒子の再配置	• 粒子の再配置を行う • ①へ戻り、同様のプロセスを繰り返す

アーキテクチャ図における位置付け



事前地図の活用方法は前頁、Octomapへの変換フローはⅢ-6-② Octomap化のデータ処理フローを参照

Source: Francisco J.Perez-Grau, Fernando Caballero, Antidio Viguria and Anibal Ollero : Multi-sensor 3D Monte Carlo Localization (MCL) for long-term aerial robot navigation, 2017

AMCLにおける重要パラメータ

変数名	内容
init_x	x軸の初期位置
init_y	y軸の初期位置
init_z	z軸の初期位置
init_a	ヨー方向の初期置
init_x_dev	初期粒子のx軸方向分散範囲
init_y_dev	初期粒子のy軸方向分散範囲
init_z_dev	初期粒子のz軸方向分散範囲
init_a_dev	初期粒子のヨー方向分散範囲
Publish_point_cloud_rate	地図の点群情報の配信レート
publish_grid_slice_rate	グリッド分割の配信レート
sensor_dev	レーザセンサーの誤差
sensor_range	レーザセンサーの測定範囲
voxel_size	ボクセルサイズ
num_particles	粒子フィルタの粒子数
odom_x_mod	x方向の有効な変動値
odom_y_mod	y方向の有効な変動値
odom_z_mod	z方向の有効な変動値
odom_a_mod	a方向の有効な変動値
resample_interval	リサンプルの頻度
update_rate	フィルタの更新頻度
d_th	x,y,zを更新する閾値
a_th	ヨー方向を更新する閾値
take_off_height	離陸を認識する高度差

Source: https://github.com/fada-catec/amcl3d/blob/kinetic/amcl3d/launch/amcl3d.launch

Visual SLAMによる自己位置推定のフロー

Visual SLAMを用いた自己位置推定の作業フローは下記の通り

① シミュレータによる事前地図の作成

- Unity上にobj形式の3D都市モデルを配置したシミュレータを構築する
- シミュレータ上の仮想カメラによって、3D都市モデルのテクスチャを撮影した画像データを取得する
- Visual SLAMのシステムに、取得した画像データを事前地図（ATLAS MAP）として保存する

② カメラ画像から特徴点を抽出

- カメラを搭載したドローンを実際に飛行させて、ストリーミング画像を取得する
- 取得した画像データに対して、画像処理による特徴点の抽出を行う（ORBアルゴリズム）

③ 事前地図との整合による自己位置推定

- 抽出した特徴点の情報をもとに、ローカル地図の作成や事前地図との照合を行い（Bag of visual word）、自己位置推定を行う（ORB SLAM3）

Ⅲ. 実証システム > 5. アルゴリズム > Visual SLAM
利用したライブラリー覧

名称	説明
ORB （Oriented FAST and Rotated BRIEF）	• 画像からの特徴点抽出および特徴量記述子の演算に用いるアルゴリズム • FASTによる特徴点の抽出（コーナー検出）とBRIEFによる特徴量記述子の考え方を組み合わせ、スケール普遍性や回転不変性のための調整を加えたもの
FAST （Features from Accelerated Segment Test）	• 特徴点抽出アルゴリズム。画像処理によって画像中に存在する角（コーナー）を検出する。 • 各ピクセルの画素値と周辺円周上の画素値を比較し、コーナーを検出する。詳細はⅢ-5 Visual SLAM ORB（Oriented FAST and Rotated BRIEF）についてを参照
Bag of visual words	• ORB SLAM3のマップ統合の検出に用いられるイメージの分類アルゴリズムの一種 • 画像情報の要素からビジュアルボキャブラリを作成し、そのボキャブラリを比較することでイメージ分類を行う手法
ORB SLAM3	• 映像情報やIMU情報を元に自己位置推定を行うVisual SLAMのアルゴリズムの1つ • 主に3つの演算スレッド（Tracking、Local Mapping、Loop & Map merging）と1つのデータベース（ATLAS）から構成されており、これらを並列処理することで、高速かつ精度の高い自己位置推定を実現 • 映像情報の画像処理には、ORBによる特徴点抽出を行う

Ⅲ. 実証システム > 5. アルゴリズム > Visual SLAM
① 3D都市モデルを活用した事前地図の作成

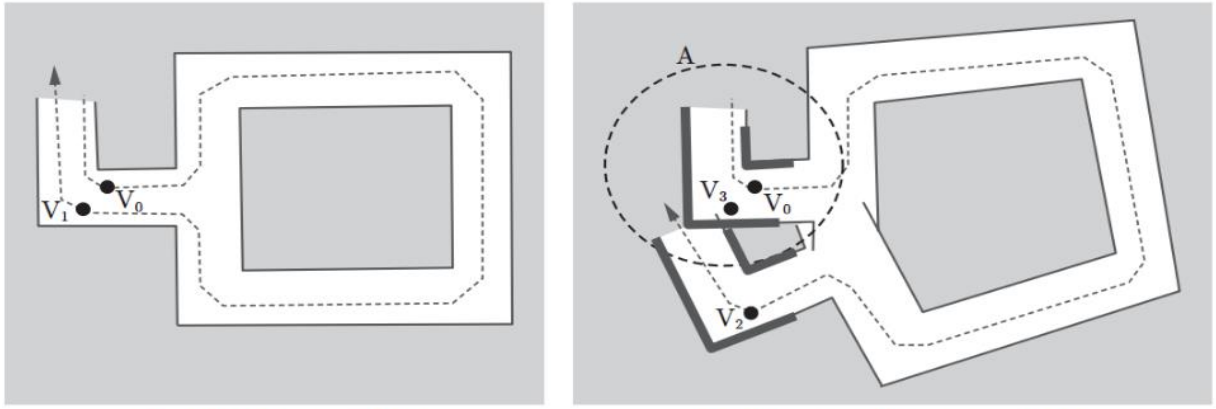
事前地図とは（再掲）

SLAMにおける課題

- 各種センサーの誤差や周辺環境の時間的な変化、推定誤差などの様々な要因により推定自己位置と周辺環境地図の両方にある程度の誤差が発生
  - 例）部屋を一周して同じ位置に戻ってきたロボットが、誤差のために初期位置と同じ場所にいるということを認識できない（下図*）

事前地図による対策アプローチ

- 自己位置と周辺環境の両方が未知であるSLAM処理のうち、ある程度確からしい事前地図（周辺環境情報）を用いることで、周辺環境マッピングの精度を上げ、全体としての精度の向上を目指す



(a) 実世界でのループ経路　(b) 地図での再訪点のマッチング

事前地図の利用方法

事前地図の利用方法

- 静的活用
  - 与えられた事前地図を真としてその事前地図を常に利用する
- 動的活用
  - 与えられた事前地図を参照しつつも実際にセンサーから得られた情報をもとに逐次地図を更新

本実証のVisual SLAMで利用する動的活用方法の特徴

- 周辺環境の時系列による変化を事前地図に反映できる
  - ある程度の原形をとどめた状態で経年的に変化している事物（壁面の局所的な損傷等）については逐次事前地図にその変化を取り込むことで常に事前地図を最新の情報に保つことができるといえる。
- 地図としての安定性としては静的な活用に劣る可能性
  - 一時的にそこにある事物についても環境地図に取り込んでしまう可能性があるため、それらがノイズとなり自己位置推定の安定性が損なわれると考えられる

② カメラ画像から特徴点を抽出

カメラストリームから得られた画像情報を元に、画像処理を行うことで特徴点の抽出を行う。

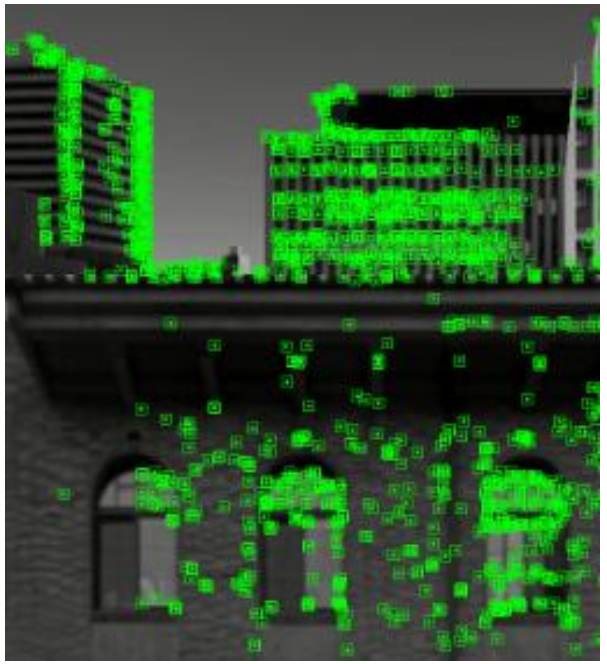
特徴点とは

- 特徴点とは、主に画像処理で用いられる用語であり、画像に含まれる特徴的な形状のことを指す
  - 点やエッジ、コーナーなどが特徴点として抽出される
  - 画像から取得された特徴点は、画像の分類やパターン認識、物体判別などの用途に用いられる

- 特徴点には特徴量記述子と呼ばれる特徴を定量化する指標を与える場合がある
  - 特徴量記述子は、違う画像の中から同じような特徴を検出することができたり、共通する特徴点のマッチングなどを行ったりすることができる

特徴点の抽出イメージ

元画像

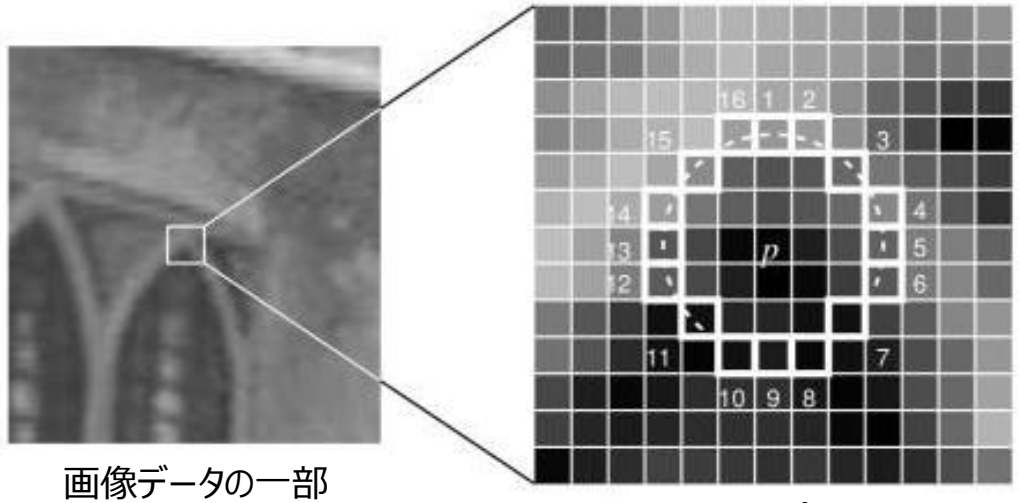


抽出された特徴点（緑点）

ORB（Oriented FAST and Rotated BRIEF）について

画像データの特徴点抽出と特徴量算出に用いるアルゴリズムで、FASTによる特徴点抽出（コーナー検出）とBRIEFによる特徴量記述子の考え方を組み合わせて、スケール不変性や回転不変性のための調整を加えた

FASTによる特徴点抽出の考え方



画像データの一部　　p周辺のピクセル

- p周辺の16画素を検査し、円上の連続するn画素の画素値が「I_p+t」より高い場合または「I_p−t」より低い場合を特徴点（コーナー）として認識する
  - p：特定のピクセル
  - I_p：pの画素値
  - t：閾値

BRIEFによる特徴量算出の考え方

比較検査パラメータ

$$\tau(\mathbf{p};\mathbf{x},\mathbf{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases},$$

特徴量

$$f_{n_d}(\mathbf{p}) := \sum_{1 \le i \le n_d} 2^{i-1}\, \tau(\mathbf{p};\mathbf{x}_i,\mathbf{y}_i)$$

- 特徴点検出エリア内における2点間の比較をn_d回行い、その結果をバイナリデータとしてまとめたものを特徴量として算出する
- BRIEFには特徴点の検出を行うアルゴリズムがないため、別途ロジック（FAST）を用いることで特徴点を検出する必要がある

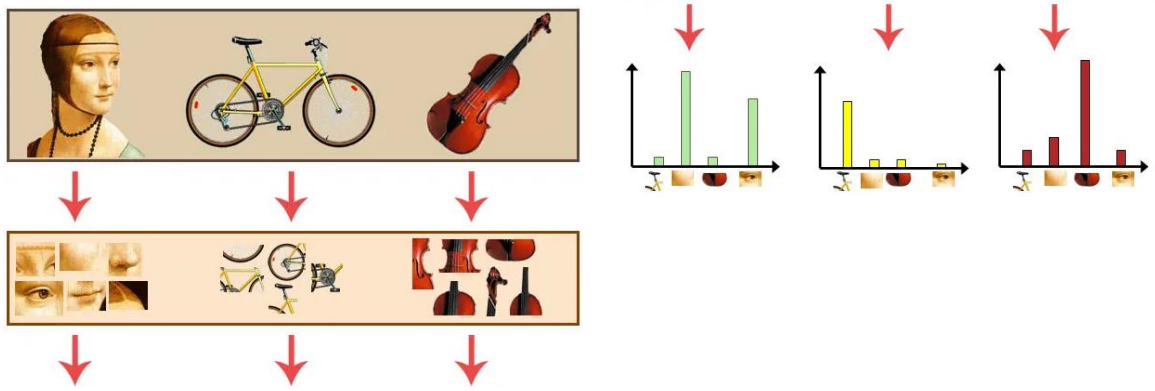
Source1: https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html (2023年2月)
Source2: Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, "BRIEF: Binary Robust Independent Elementary Features", 11th European Conference on Computer Vision (ECCV), Heraklion, Crete. LNCS Springer, September 2010.

③ 事前地図との整合による自己位置推定

Visual SLAMにおいては、画像処理を用いて複数の画像から同一箇所の検知を行うケースが多い。検知された同一箇所情報を元に、ループとじ込み処理や事前地図との照合などの処理を行う。

画像の分類と同一箇所検知

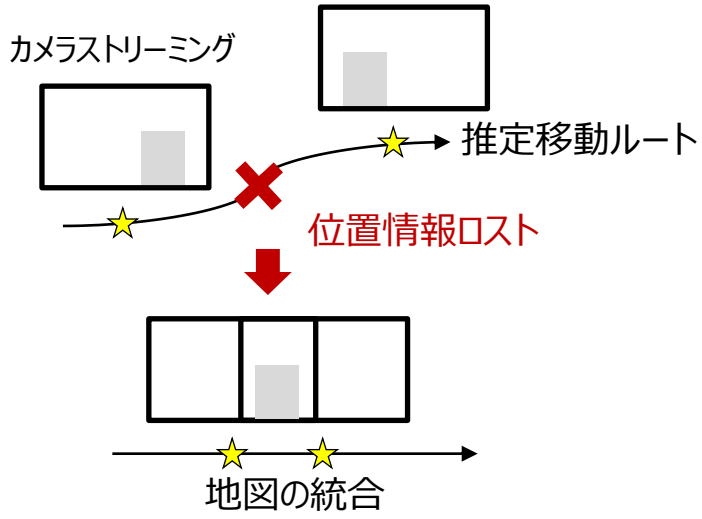
- VisualSLAMでは自己位置推定のために、別の画像から同一箇所を検出する必要がある
- 本実証では同一箇所の検出にBag of Visual Wordsを利用する
  - Bag of Visual Wordsは、自然言語における文章の分類に用いられるBag of Wordsを画像処理に応用したもの
    - Bag of Wordsは順番を考慮せず、文章の中で発生する単語の頻度分布から単語を分類するものである
  - 画像を特徴点やその特徴量記述子のデータセットとみなし、その出現頻度をもとに類似画像の検索や画像の分類が可能

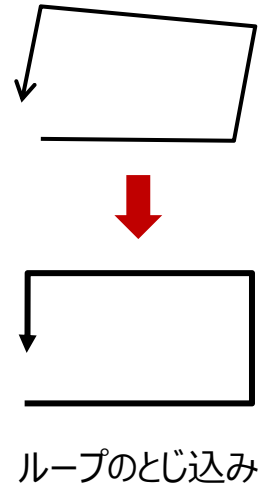


Bag of Visual Wordsにおける出現頻度分布作成イメージ *1

ループとじ込みと地図の統合

- **地図の統合**
  - 何らかの要因で一時的に画像による位置推定が難しくなった場合、その間の位置推定精度は急激に低下するが、事前地図によって同一箇所の検知ができると一貫した自己位置推定が可能となる
- **ループのとじ込み**
  - 一般にSLAMシステムは誤差の蓄積より位置推定や地図の歪みが発生するが、特にループなどではこの誤差により同一箇所の検出が困難となる
  - 事前地図があることで、同一箇所の検知およびループとじ込み処理を行うことができ、こういった歪みを修正することができる





* Source1: https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb (2023年2月)

Bag of visual wordsについて

現在のフレームとATLAS MAPに格納されているキーフレームとを照合することで地図の統合を行う際に用いるアルゴリズムで、画像情報の要素からビジュアルボキャブラリを作成し、そのボキャブラリを比較することでイメージ分類を行う

ATLAS MAPとの照合

類似性スコア

$$s(\mathbf{v}_1, \mathbf{v}_2) = 1 - \frac{1}{2}\left|\frac{\mathbf{v}_1}{|\mathbf{v}_1|} - \frac{\mathbf{v}_2}{|\mathbf{v}_2|}\right|$$

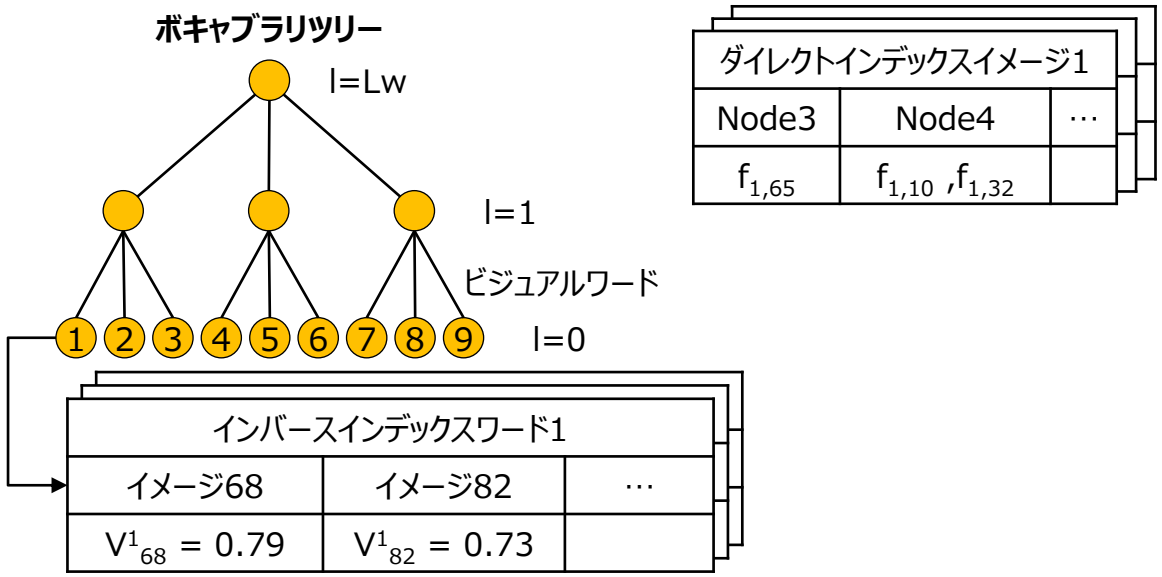
正規化類似性スコア

$$\eta(\mathbf{v}_t, \mathbf{v}_{t_j}) = \frac{s(\mathbf{v}_t, \mathbf{v}_{t_j})}{s(\mathbf{v}_t, \mathbf{v}_{t-\Delta t})}$$

特徴点抽出　特徴量の算出　クラスタリング　ボキャブラリの定義

- ビジュアルボキャブラリとは、既存のデータセットからORBによる特徴点検出と特徴量算出を実施した結果をもとにk-means法によるクラスタリングを行って抽出された特徴のこと
- ビジュアルボキャブラリツリーをもとに、画像データからBag of wordベクトルを算出し、上記で定義される「類似性スコア」からデータベースとの照合を行う

ボキャブラリツリーの構造



- ボキャブラリツリーは、ビジュアルボキャブラリの作成手法を階層的に行うことで作成される

Source1: D. G´alvez-L´opez and J. D. Tard´os, “Bags of binary words for fast place recognition in image sequences,” IEEE Transactions on Robotics vol. 28, no. 5, pp. 1188–1197, 2012.
Source2: https://jp.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html（2023年2月）

ORB SLAM3について

自己位置推定の精度と速度を上げるために、主に3つの演算スレッドと1つのデータセット（ATLAS）から構成されるORB SLAM3というアルゴリズムを採用

3つの演算スレッド概要

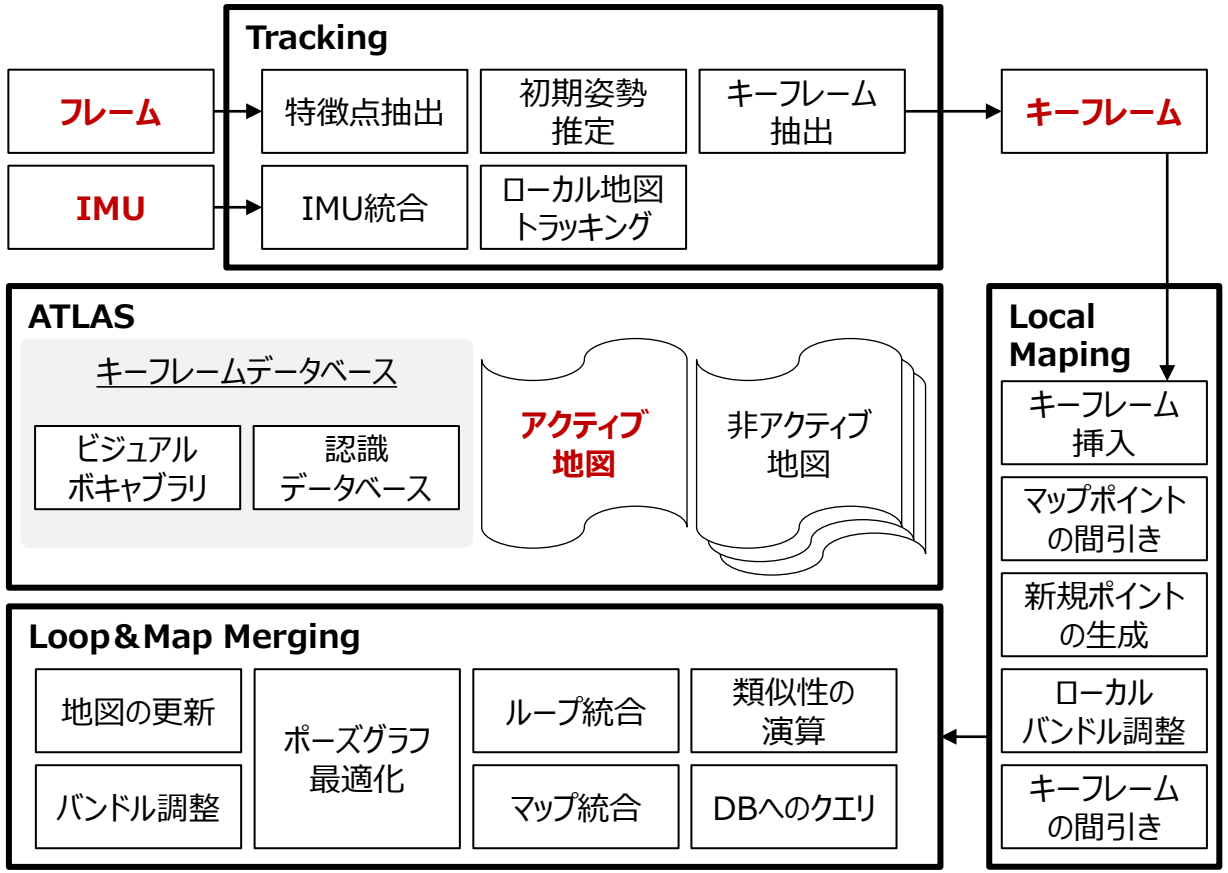
- **「Tracking」・「Local Mapping」・「Loop＆Map Merging」**の3つの演算スレッドを、並列的に処理する

- **「Tracking」**
  - アクティブ地図をもとに、現在のフレームにおける位置と姿勢を推定するとともに、キーフレームの抽出を行う
  - アクティブ地図から自己位置をロストした場合は、ATLAS上の地図セットを検索し、アクティブ地図を切り替える

- **「Local Mapping」**
  - キーフレームをアクティブ地図に挿入する
  - 現在のフレームに近接するキーフレームの範囲内で、冗長なデータの削減およびバンドル調整を行い、ローカル地図を改善する

- **「Loop & Map Merging」**
  - 現在のフレームとアクティブ地図内およびATLAS上の他のマップのキーフレームを比較し、共通の箇所を検出する
  - アクティブ地図内で同一箇所の検出をした場合は、Loop Correctionを行い、他の地図内に同一箇所を検出した場合は地図の統合を行う

ORB SLAM3の構成図



Source: Carlos Campos, et al.: ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM, IEEE Transactions on Robotics 2021 Dec., Vol 37, Num 6, P.1874-1890をもとに作成

ORB SLAM3における重要パラメータ

変数名	内容
Camera.type	カメラの種類（Pinholeなど）
Camera1.fx	カメラの焦点距離（x方向）
Camera1.fy	カメラの焦点距離（y方向）
Camera1.cx	カメラの光学中心（x方向）
Camera1.cy	カメラの光学中心（y方向）
Camera1.k1	カメラの半径方向歪み補正パラメータ1
Camera1.k2	カメラの半径方向歪み補正パラメータ2
Camera.p1	カメラの円周方向歪み補正パラメータ1
Camera.p2	カメラの円周方向歪み補正パラメータ2
Camera.width	カメラ画像の横幅
Camera.height	カメラ画像の縦幅
Camera.fps	1秒当たりのフレーム数
Camera.RGB	画像の色定義（RGB or BGR）
ORBextractor.nFeatures	画像あたりの特徴点の最大数
ORBextractor.scaleFactor	スケールピラミッドの階層ごとの縮小率
ORBextractor.nLevels	スケールピラミッドの階層数
ORBextractor.iniThFAST	特徴点抽出に用いる閾値の初期値
ORBextractor.minThFAST	特徴点抽出に用いる閾値の最小値

3D都市モデル一覧

地物	地物型	属性区分	属性名	内容
建築物LOD2	bldg:Building	空間属性	bldg:lod2Solid	建築物のLOD2の立体

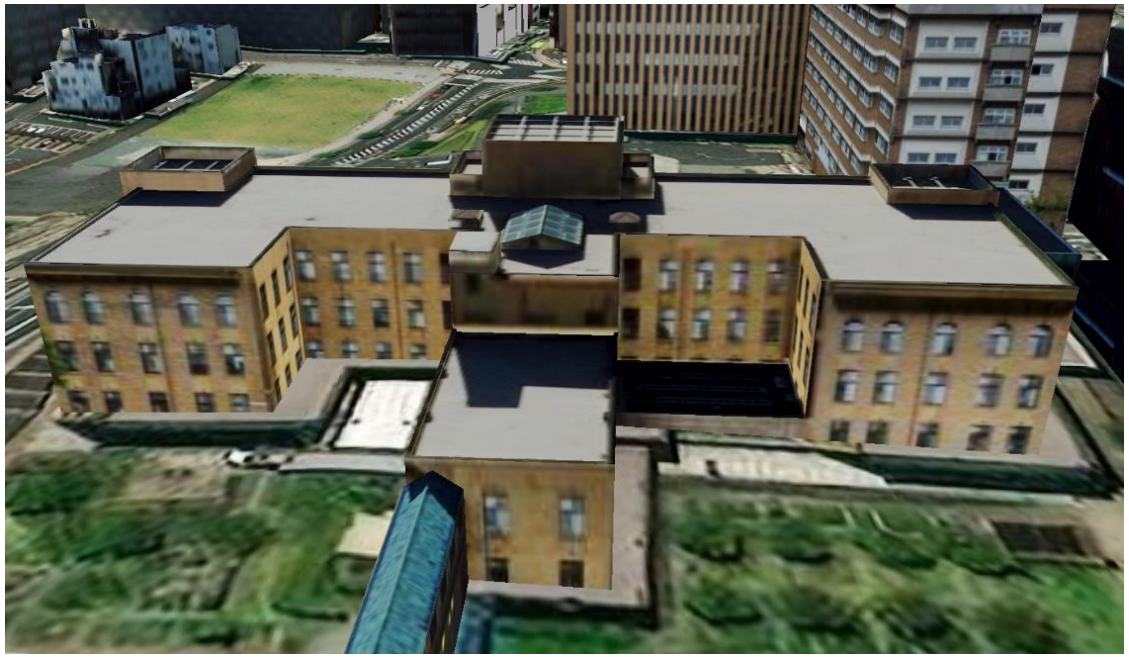
3D都市モデルのテクスチャ

本実証ではVisual SLAMにおいてテクスチャに高い解像度が求められると予想されたため、事前に建物壁面の高解像度データを取得し、高解像度のテクスチャを張り付けた3D都市モデルを作成した

通常のテクスチャ（航空測量）

- 一般的な方法で作成された3D都市モデルのテクスチャ

地上から撮影したテクスチャ



- 地上から撮影した高解像度の写真を画像データとして取り込み、3D都市モデルの建物壁面に貼り付け

※いずれも飛行エリアとなる山梨県庁の3D都市モデル

活用データ一覧：その他

本実証における3D都市モデル以外の活用データは下記の通り

活用データ	内容	データ形式	出所
点群データ	LiDARセンサーで取得した点群データ	バイナリ	LiDARセンサー
画像データ	ステレオカメラで取得した画像データ		ステレオカメラ
距離データ	ToFセンサーで取得した距離情報データ		ToFセンサー
姿勢データ	IMUで取得した機体・センサーの姿勢データ		IMU
GNSS位置情報データ	GNSSモジュール（RTK-GNSSも含む）で取得したGNSS位置情報データ		RTK-GNSSモジュール
テレメトリ（ドローン機体情報）	位置情報や高度、センサーのデータ、バッテリー状況等、ドローンの飛行状態を表すパラメータ群	MAVLink	フライトコントローラ

センサーデータ（1/2）

LiDARセンサー、ステレオカメラの詳細は下記の通り

名称	LiDARセンサー	ステレオカメラ
イメージ画像		
メーカー	Ouster社	Intel社
モデル・型式	OS-1 128	Real sense d435i
選定理由	• 2022年時点において、標準的なドローンに搭載可能なサイズ・重量のLiDARセンサーとしては、取得点群の点数や視野角・精度などが非常に高い水準にあるため	• ドローンに搭載可能なステレオカメラのうち、性能や機能に対して重量・サイズが小さく、搭載カスタマイズの柔軟性や航行時間が十分に確保できると考えられるため • 本実証で用いるORB SLAM3ライブラリのチュートリアル等、ドキュメントが充実しており、実装の時間を短縮できるため • IMUを内蔵しており、別途IMUを搭載する必要がないため
仕様詳細	• 視野：45° • 範囲：90m（10%）、200m（最大） • 解像度のチャネル：128 • 重量：447g • 精度：±0.5cm	• 出力解像度：1920 x 1080 @30fps • 画角：69° x 42° • 深度精度：<2% up to 2m • サイズ：90mm x 25mm x 25mm
対象システム	• LiDAR SLAM	• Visual SLAM

センサーデータ（2/2）

ToFセンサー、IMU、RTK-GNSSの仕様は下記の通り

名称	ToFセンサー	IMU	RTK-GNSSモジュール
イメージ画像			
メーカー	Terabee社	VECTORNAV社	Hex社
モデル･型式	TeraRanger Evo 60m	VN-100	Here 2+RTK GPS受信機
選定理由	• ドローンへの搭載を前提として設計された軽量なセンサーとなっているため • 過去にドローンへの実装実績があり、取り付け工数を削減できるため	• 軽量かつ消費電力も高くないため • 別機材ではあるもののLiDARセンサーとのデータ統合の実績があるため	• フライトコントローラとの互換性があり、ドローンとの物理的接続や設定が容易であるため • ドローンのRTK-GNSSモジュールとして幅広く使われており、ドキュメントが充実しているため
仕様詳細	• センサー形式：赤外線 • 計測範囲：0.5 ~ 60m • 測定レート：240 per second • 解像度：0.5cm (14m以内)、2 cm (14m以上) • 精度：±4cm（14m以内）、1.5% (14m以上)	• ジャイロバイアス安定性：5~7°/hr • 加速度バイアス安定性：<0.04 mg • オンボードカルマンフィルタ更新レート：400Hz • サイズ：36mm x 33mm x 9mm • ピッチ/ロール精度：0.5°	• ICM20948電磁コンパス内蔵 • 72チャンネル ublox M8Nエンジン • 最大更新レート：10Hz • 位置精度：3D Fix
対象システム	• LiDAR SLAM、Visual SLAM	• LiDAR SLAM	• LiDAR SLAM、Visual SLAM

テレメトリ（ドローン機体情報）

ドローン飛行中に転送されるテレメトリ情報は各種センサーデータをはじめ多岐にわたっており、代表例は下記の通り

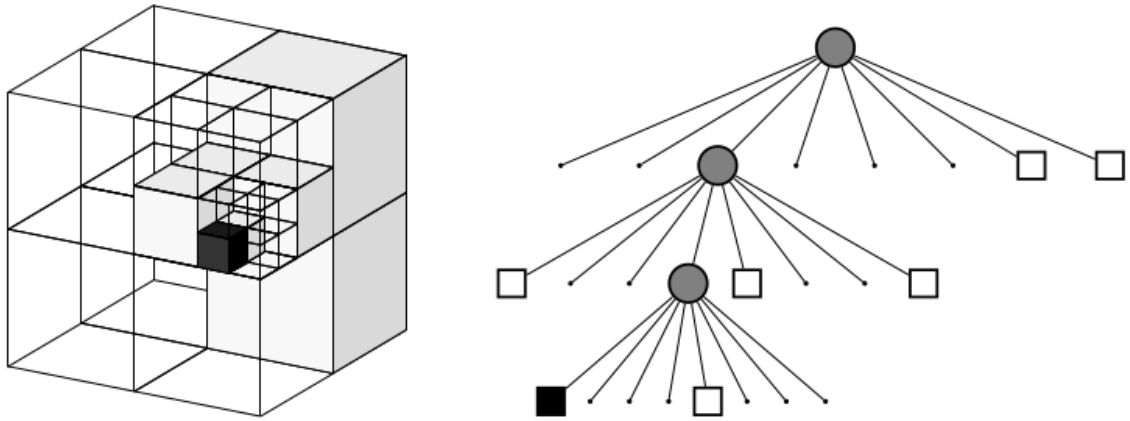
変数名	内容
Bat voltage	電池の電圧
Accel x	IMUのx方向加速度
Accel y	IMUのy方向加速度
Accel z	IMUのz方向加速度
Gyro x	ジャイロのx方向読み値
Gyro y	ジャイロのy方向読み値
Gyro z	ジャイロのz方向読み値
Gps status	GNSSのfix状況
Sat Count	GNSSが使用している衛星数
Latitude	GNSSの取得した緯度
Longitude	GNSSの取得した経度
GPS HDOP	GNSSの水平方向精度
Altitude	機体高度
Ground Speed	対地速度

データ処理一覧

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）
Octomap	LiDAR SLAMにおける事前地図	• 必要な領域の切り取りおよびメッシュの最適化 • 点群データへの変換 • Octomap形式への変換	Blender ROS PLCパッケージ ROS Octomapパッケージ	3Dモデルデータ（obj形式）
ATLAS MAP	Visual SLAMにおける事前地図	• 仮想空間上にテクスチャ付きの3D都市モデルを配置し、仮想カメラによる動画データ取得を行う • えられた動画データを用いてVisual Slamシステムを稼働し、環境地図を得る	Unity ROS ORB slam2パッケージ	3Dモデルデータ（obj形式）

Octomapとは

Octomapの概要 *



- Octomapとは、八分木構造を応用した3次元占有グリッドマップ
- Octomap形式は、効率的に空間構造を表現できるため、SLAMのような高い演算リソースが求められる処理に活用されるデータ形式
- Octomap形式の構造概念
  - あるグリッドに対して占有率が十分多い場合：物体があるとみなす
  - あるグリッドに対して占有率が十分少ない場合：物体はないとみなす
  - あるグリッドに対して占有率が中間値の場合：そのグリッドをさらに8分割して同様の手続きを繰り返す

Octomapの特徴

- フル3Dモデル

形状に依存せず、任意の環境においてモデル化が可能。フリースペースと占有されているスペースに加え、未知スペースを設定することができるため、自動探索ロボットなどの未知領域を動くロボットのナビゲーションへの応用に適している。

- 動的に更新可能

事前に準備している情報に加え、新たにセンサーにより取得した情報を動的に付加することが可能。また、モデリングや更新に際しては、確率論的なアプローチを採用しており、センサーノイズや環境の動的な変化も考慮したんモデリングができる。

- 柔軟性

モデルの拡張は事前に設定する必要がなく、必要に応じて動的に拡張することができる。また、地図は複数の解像度の情報を階層的に持っているため、状況や必要に応じて適切な解像度の地図を用いることで効率的なリソース活用ができる。

- コンパクト

解像度が一律ではなく、必要な場所のみに高い解像度が割り当てられるため、メモリ・ストレージそれぞれの観点から効率的なデータの利用が可能。

*: Robotics with ROS 公式サイト(2023年2月)(https://ros-developer.com/2017/11/27/octomap-explanierend/)

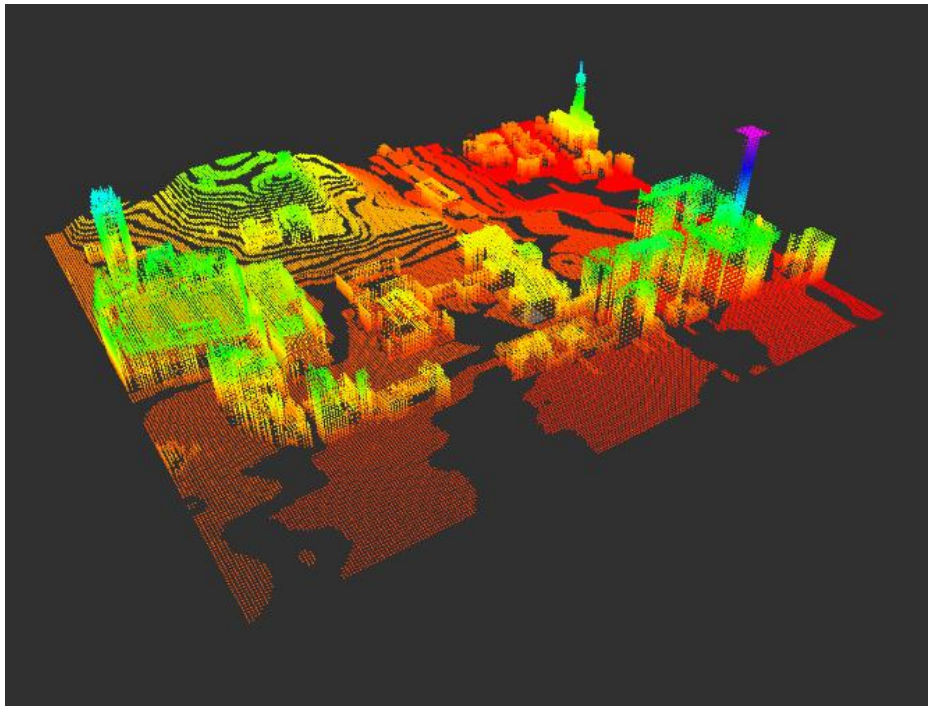
Octomap化のデータ処理フロー

Octomap形式にデータを変換処理した実証対象エリアの3D都市モデルを事前地図として活用する

3D都市モデルのデータ変換フロー

データ変換1 （CityGML→obj）	• 対象エリアの3D都市モデルをCityGML形式→obj形式に変換する（FME Desktop）
領域指定と メッシュ適性化 （obj形式）	• obj形式に変換した3D都市モデルから必要な領域の指定とメッシュの適性化を行う（Blender）
データ変換2 （obj→pcd）	• obj形式→pcd形式に変換する（ROS PLC）
データ変換3 （pcd→Octomap）	• pcd形式→Octomap形式に変換する（ROS OctoMap）

Octomapのビジュアルイメージ



飛行対象エリアとなる山梨県庁周辺のOctomap

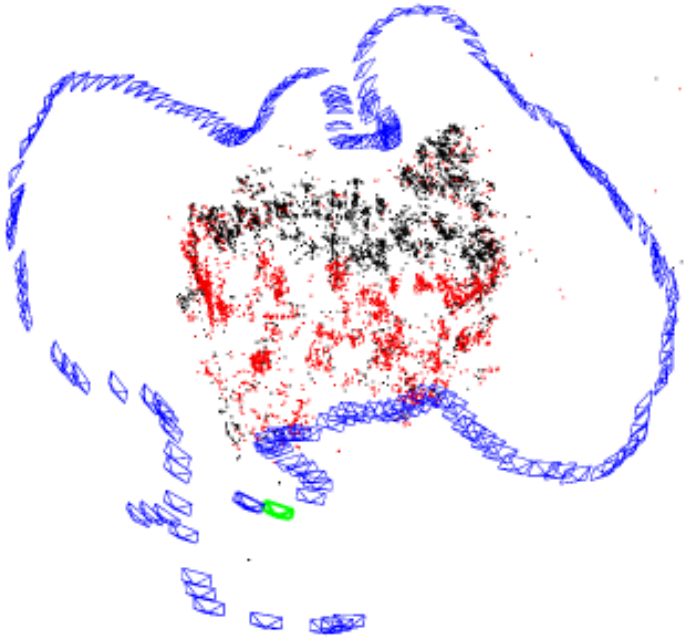
ATLAS MAPとは

ATLAS MAP概要

パラメータ名	内容
3D position	各特徴点の3次元位置
視覚方向	各特徴点をとらえることができる視覚方向
ORB特徴量	各特徴点の特徴量
カメラ姿勢	キーフレームおけるカメラの姿勢
カメラの内部パラメータ	焦点距離などのカメラの内部パラメータ

- ATLAS MAPとは、本実証においてVisual SLAMのメインライブラリとして使用するORB SLAM3で事前地図データとして用いられるデータ構造
- 各データは断片的な地図情報の集積として保存されているが、別の地図上に同一箇所（データ）を検出することにより、1つの地図として統合される
- 上記の表は、ATRAS MAPとして保存されるデータの一部

ATLAS MAPのビジュアルイメージ *



青：キーフレーム、 緑：現在のカメラ位置、 黒：特徴点の3D位置
赤：現在のローカルマップに属する特徴点の3D位置、

*: Raúl Mur-Artal, J. M. M. Montiel and Juan D. Tardós, “ORB-SLAM: a Versatile and Accurate Monocular SLAM System” IEEE Transactions on Robotics vol. 31, no. 5, pp. 1147–1163, 2015.

ATLAS MAPのデータ処理フロー

Unity上において、obj形式にデータ変換した実証対象エリアの3D都市モデルを配置したシミュレータを構築し、シミュレーションにより取得した画像ストリームデータを事前地図として活用する

事前地図の作成フロー

データ変換（obj形式）

- 3D都市モデルをCityGML形式→obj形式に変換する（FME Desktop）

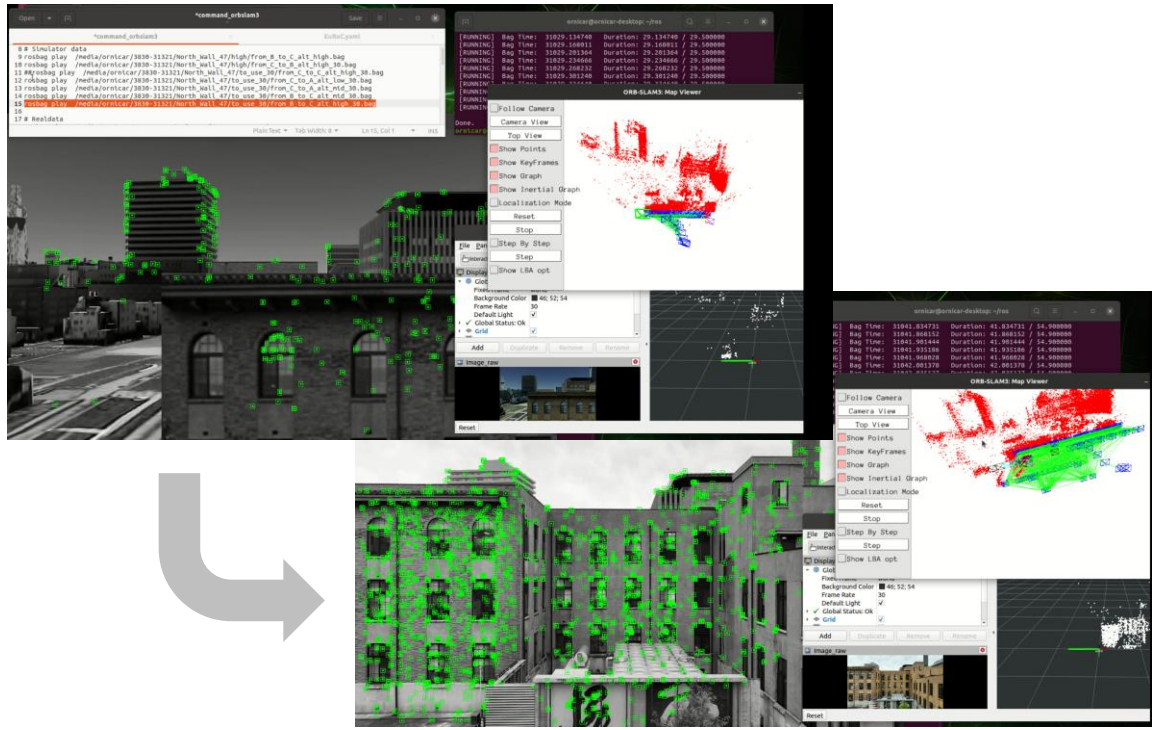
画像ストリームデータ取得

- Unityを使い、3D都市モデルを配置したシミュレータを構築する
- シミュレータにより仮想カメラの画像データを取得する

ATLAS MAP（環境地図）作成

- ROS ORB SLAM3を用いて、Visual SLAMにおいて画像データから得られた特徴点や情報等をまとめて、事前地図を作成する

事前地図作成のビジュアルイメージ



左上：シミュレーションによる事前地図作成、右下：カメラ画像との照合

Source: Carlos Campos, et al.: ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial and Multi-Map SLAM, IEEE Transactions on Robotics 2021 Dec., Vol 37, Num 6, P.1874-1890

出力データ一覧

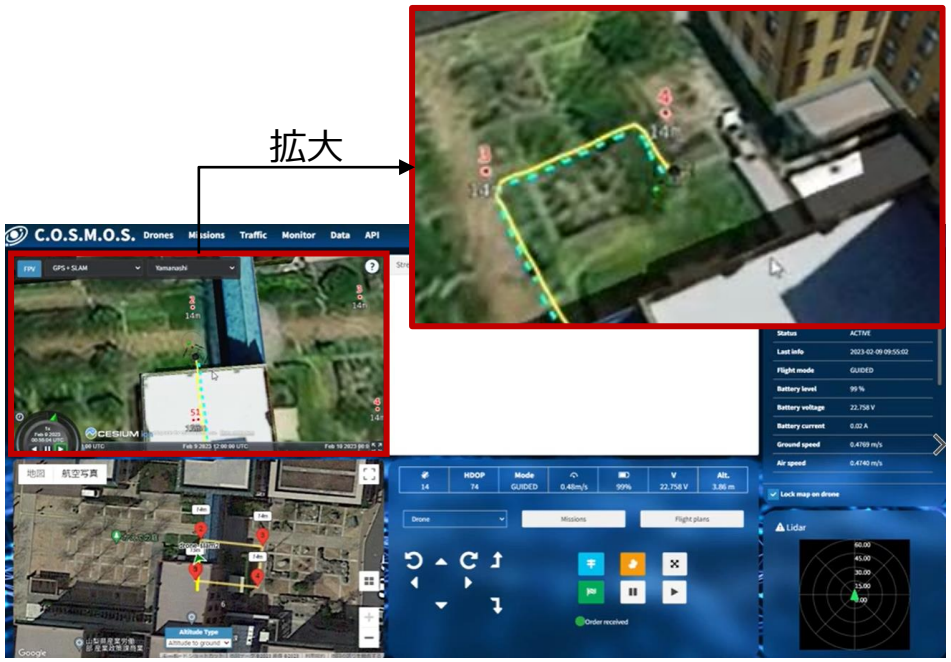
本実証における出力データは下記の通り

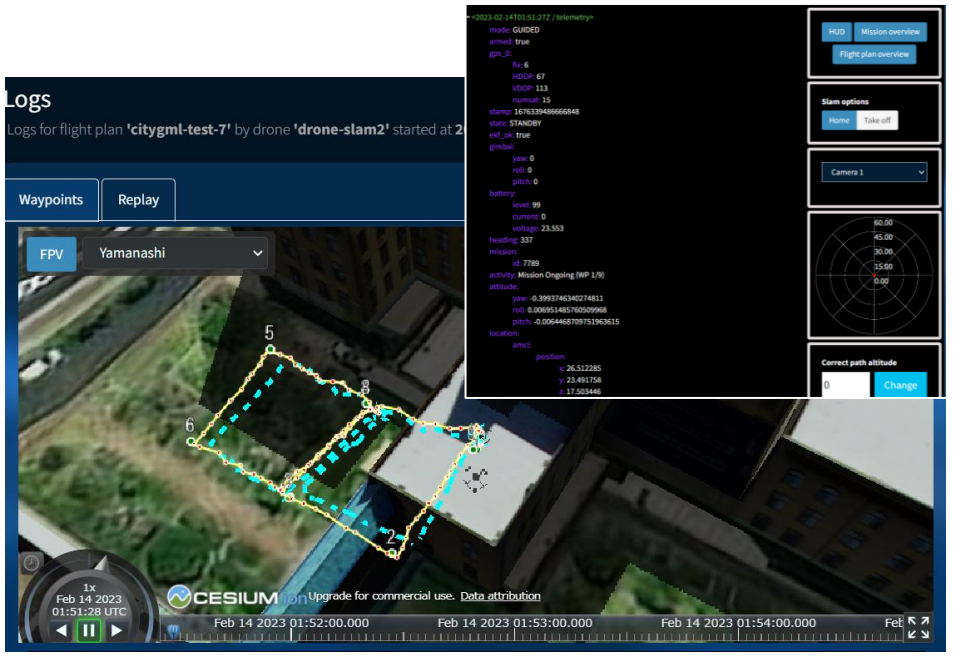
出力データ	内容	データ形式
自己位置情報	離陸地点を基準とした自己位置の座標（x,y,z）	－
障害物回避飛行ルート	障害物を考慮した飛行ルート	MAVLink

UI/UXの全体フロー

運航管理システムのCOSMOSにおいて、ドローン飛行監視画面を確認した上で実際にドローンを飛行させて、飛行ルートのログを確認する

運航管理システムにログイン → ドローン飛行監視画面の表示・確認 → 飛行ルートのログ確認

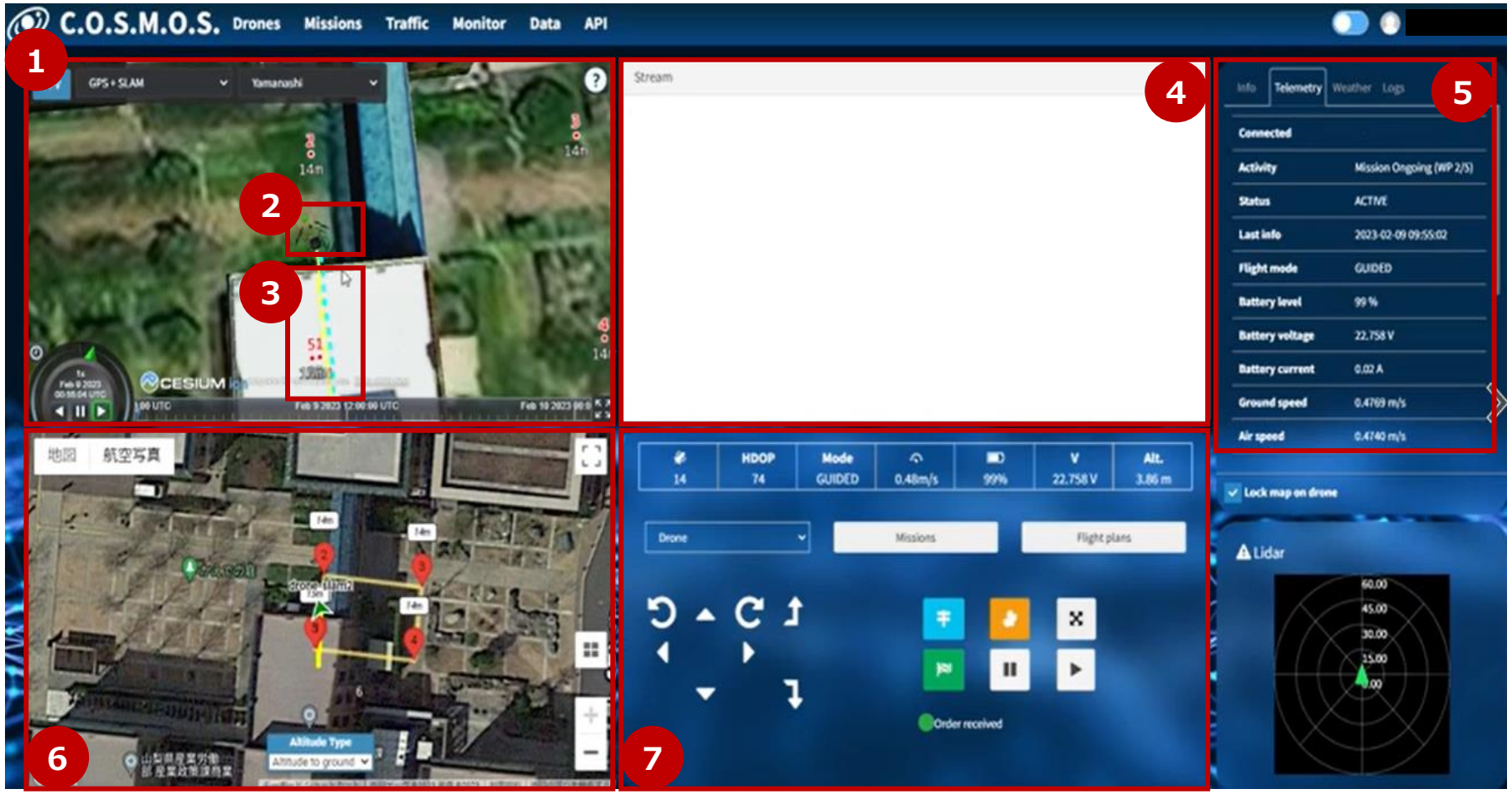




- PC上にて、運航管理システムのCOSMOSへログインする
- ドローン飛行監視画面を表示して、ドローンの実飛行を実施
- 3D都市モデルを用いた3D地図表示にて飛行ルートを確認
- センサーやカメラ等から取得した飛行ルートのログを確認
- 黄色線：GPS（RTK）、青色点線：各SLAM

COSMOS（ドローン監視画面）

運航管理システムのCOSMOSにおけるドローン監視画面のWebUI詳細は下記の通り



#	機能名	説明
①	3D地図表示	3D都市モデルを活用した地図を表示できる
②	機体位置表示	機体の位置を黒丸印で表示できる
③	飛行ルート表示	ドローンの飛行ルートを表示できる • 黄：GPS（RTK） • 青：各SLAM
④	ストリーミング表示	ドローンカメラの映像を表示（本実証では未使用）
⑤	機体情報詳細表示	ドローンの機体情報を表示できる
⑥	2D地図表示	Google Mapの2D地図を表示できる
⑦	遠隔操作パネル	遠隔操作パネルを表示できる

システムテスト結果一覧

システムテストの項目と結果は下記の通り

試験項目	確認内容	結果
3D都市モデルとドローン位置の表示	• 実際に該当ドローンを当該地区で電源を入れ、3D地図上に表示されるか	合格
LiDAR SLAMによる自己位置推定	• Rviz（ROS用の可視化ソフトウェア）を用いてリアルタイムで取得点群と3D都市モデルによる事前地図の位置関係を表示した状態でドローンを飛行させ、自己位置推定が機能するか	合格
LiDAR SLAMとGNSSの結果の可視化	• LiDAR SLAMで実際に飛行を行い、GNSSとVisual SLAMの結果がそれぞれ3D地図上で表示されるか	合格
3D都市モデルを利用したVisual SLAM用の事前マップの設定	• シミュレーションで取得した画像をもとにVisual SLAM演算を行い、ファイルとして保存されるか	合格
Visual SLAMによる自己位置推定	• 保存した事前マップを使って同一箇所で撮影した画像を用いてVisual SLAM演算を行い、自己位置推定ができるか	合格
Visual SLAMとGNSSの結果の可視化	• Visual SLAMで実際に飛行を行い、GNSSとVisual SLAMの結果がそれぞれ3D地図上で表示されるか	合格

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．実証システム

Ⅳ．実証技術の検証

Ⅴ．成果と課題

Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
①検証内容｜概要

目的	**LiDAR SLAM/3D都市モデル×エッジ処理×Shared Computingの検証** • LiDAR SLAM/3D都市モデル×エッジ処理とLiDAR SLAM/3D都市モデル×Shared Computingの構成パターンで、どの程度まで自己位置測定の精度を高く推定できるか検証する • Shared Computingを用いることやインターネット通信の速度が、どの程度ボトルネックとなるか確認する
実施期間	2023年2月6日～10日
実施場所	山梨県甲府市
主な参加者	現場統括、パイロット、サブパイロット、ソフト・ハードエンジニア、現場進行：各1人
実施内容	• LiDARセンサーおよび周辺の3D都市モデルをインストールしたコンピュータを搭載したドローンを、山梨県庁敷地内で飛行させ、LiDAR SLAMによる自己位置推定を行う • 自己位置推定の結果と、同時に計測を行うRTKを比較して自己位置推定精度を検証する • SLAMの演算処理を機体側で行った場合とサーバー側で行った場合を比較して、サーバー側で演算処理を行う場合のシステムの有用性を検証する • 自己位置推定精度検証およびサーバー演算の有用性検証から今後の課題を抽出する

①検証内容｜検証方法の詳細

LiDAR SLAMとVisual SLAMにおいて、2パターンの事前地図と4パターンの演算処理によるドローン飛行結果から、それぞれの自己位置推定精度を測定して比較検証する

検証内容

項目	内容
検証目的	LiDAR SLAMによる自己位置推定の精度検証
検証方法	• 精度が数cmレベルのRTKを真値と仮定して、4つのシナリオを使いLiDAR SLAMの位置精度を評価する（ログ分析） - 全球測位衛星システム（GPS）での誤差（～5m以内）を実用レベルの精度と仮定して、これを閾値に本SLAMシステムの有用性を評価する

検証シナリオ

シナリオ	事前地図	演算処理
No.1	3D都市モデル	エッジ（機体）
No.2	LiDARセンサーによる事前取得点群データ	
No.3	3D都市モデル	シェアードコンピューティング（サーバー）
No.4	LiDARセンサーによる事前取得点群データ	

Ⅳ. 実証技術の検証 ＞ 1. LiDAR SLAMによる自己位置推定の精度検証

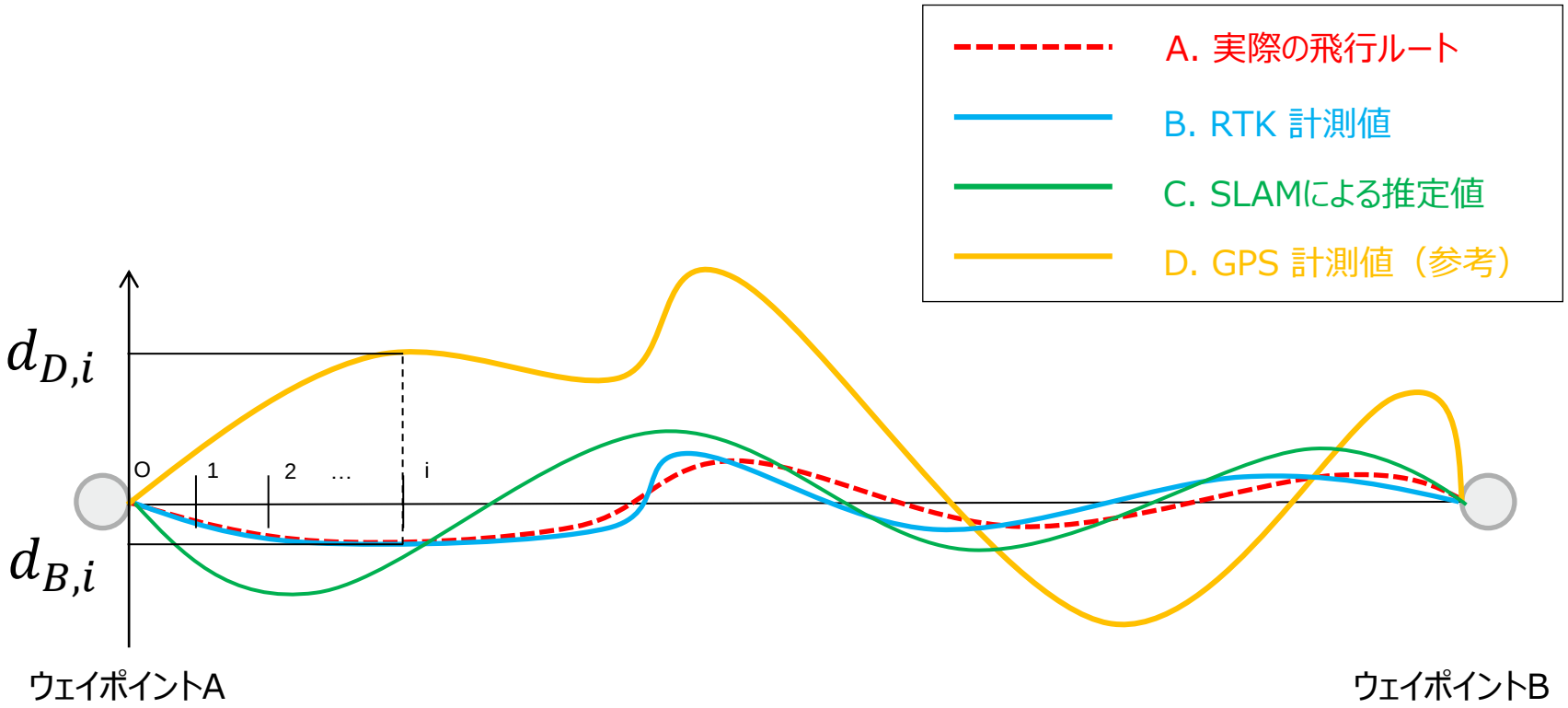
①検証内容｜精度の評価方法

実際の飛行ルートをRTKの計測値と限りなく近いものと仮定し、SLAMによる推定値との誤差頻度を検証する

検証内容

- 「A.実際の飛行ルート」は未知（計測不可）であるため、これを計算することはできない。したがって、実際の自動航行および手動航行におけるドローンの飛行ルートは十分に「B.RTK 計測値」に近いと仮定する
- 各手法のサンプリング点iでの位置推定の誤差は下記で表現される
  SLAMによる推定値
  ： $|d_{D,i} - d_{B,i}|$
- このような想定の下、この誤差の頻度分布を作成し、飛行中にどの程度の誤差が発生していたのか検証する

検証イメージ



Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証

①検証内容｜フライトルート

以下8つのフライトルートを設定し検証する

：検証エリア
：航行ルート

フライトA

- 建物に対して水平・垂直方向のルート
- 壁面に対して30m程度の距離の往復でどの程度の精度が出るのかを確認する

フライトB

- カーブを含むルート
- 壁面に対して水平・垂直以外の動線が入るルートで自己位置推定に影響が出るのかどうかを確認する

Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜サマリ

シナリオ	精度平均値（m）	標準偏差（m）	示唆
No.1 3D都市モデル エッジ処理	フライトA：0.164 フライトB：0.148	フライトA：0.178 フライトB：0.086	3D都市モデルとLiDAR SLAMを組み合わせることで、一般的なGNSSを用いた場合と比較しても精度良く自己位置推定が可能であることが実証することができた
No.2 点群データ エッジ処理	フライトA：0.100 フライトB：0.143	フライトA：0.167 フライトB：0.074	事前取得したデータを用いた場合の精度は3D都市モデルを用いた場合と同等程度であった。このことから、LiDAR SLAMを用いようとしたとき、3D都市モデルを用いることで事前データの取得の工数を大幅に削減できる可能性が示された
No.3 3D都市モデル シェアードコンピューティング処理	フライトA：0.235 フライトB：0.277	フライトA：0.188 フライトB：0.152	SLAM処理をエッジで行った場合とリモートで行った場合での精度についての有意な差はみられなかった。このことから通信を用いることでの精度に対する大きな影響はなく、前述の回線速度の問題が改善されればクラウド環境で演算を行うというコンセプトは実現可能と考えられる
No.4 点群データ シェアードコンピューティング処理	フライトA：0.152 フライトB：0.166	フライトA：0.076 フライトB：0.067	No1とNo2の比較と同様に、リモート演算環境においても3D都市モデルを用いた場合と事前取得したデータを用いた場合の結果に大きな差異は見られなかった

Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜精度検証結果

シナリオ	フライトA (m)				フライトB (m)			
	最大	最小	平均	標準偏差	最大	最小	平均	標準偏差
No.1 3D都市モデル エッジ処理	0.893	0.005	0.164	0.178	0.404	0.022	0.148	0.086
No.2 点群データ エッジ処理	0.846	0.006	0.100	0.167	0.309	0.017	0.143	0.074
No.3 3D都市モデル シェアードコンピューティング処理	0.683	0.001	0.235	0.188	0.603	0.022	0.277	0.152
No.4 点群データ シェアードコンピューティング処理	0.360	0.012	0.152	0.076	0.309	0.017	0.166	0.067

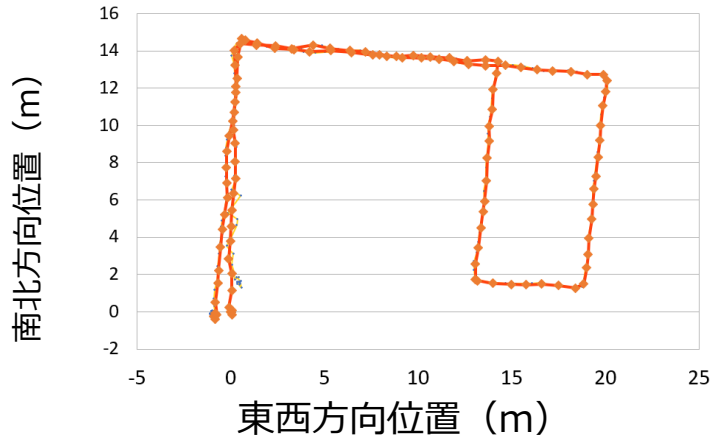
Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜No.1 3D都市モデル、エッジ処理

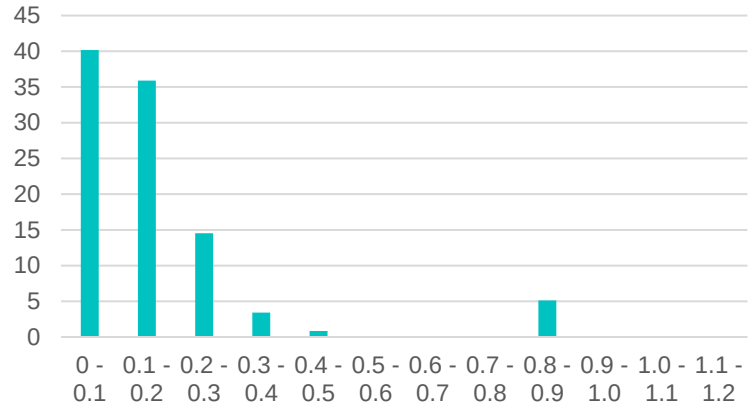
SLAMとRTKによる位置予測

SLAM:青
RTK：オレンジ

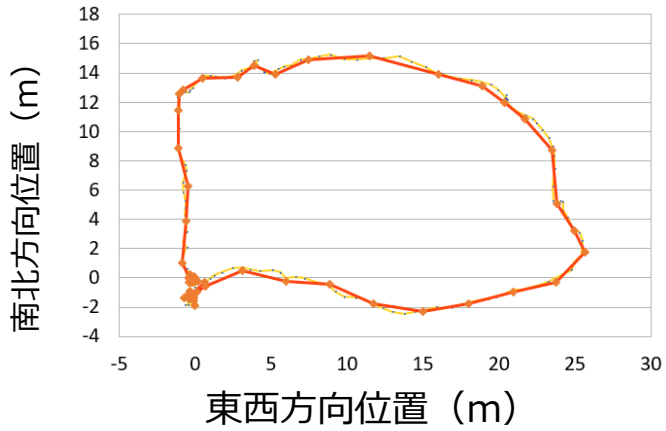
RTKとの差異の検出割合（%）

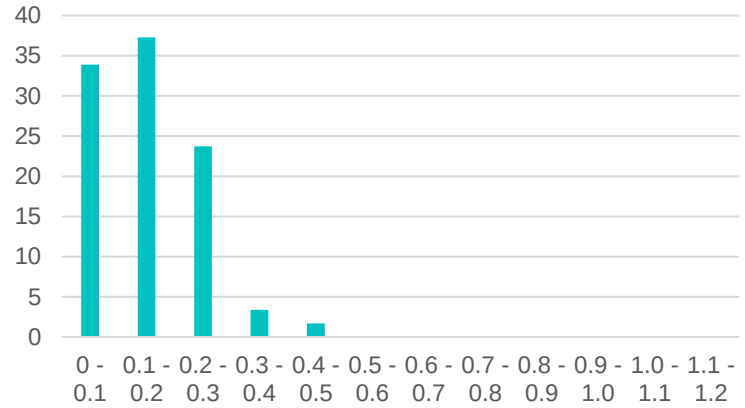
フライトA





フライトB





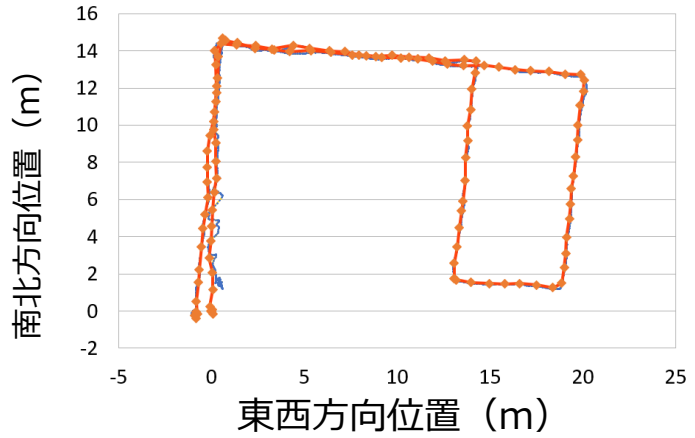
Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜No.2 点群データ、エッジ処理

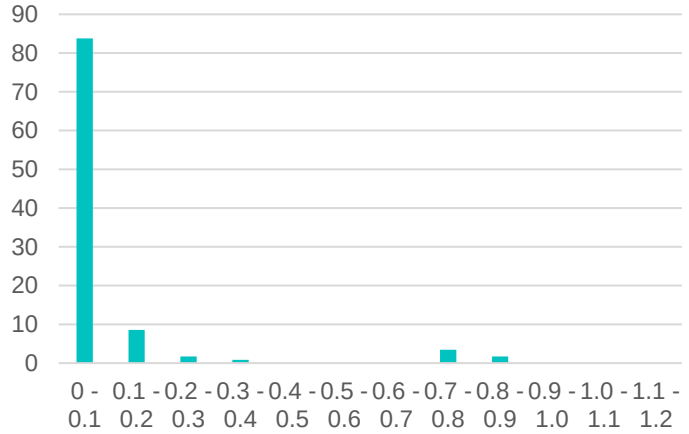
SLAMとRTKによる位置予測

SLAM:青
RTK：オレンジ

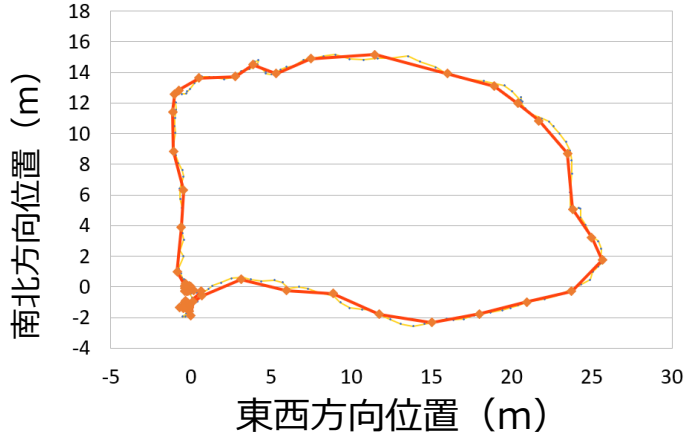
RTKとの差異の検出割合（%）

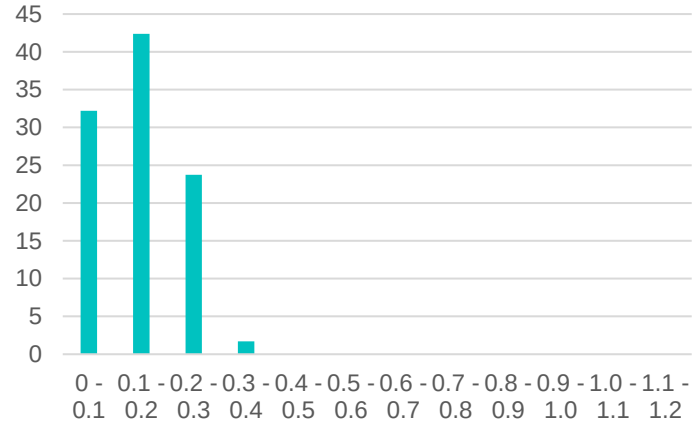
フライトA





フライトB





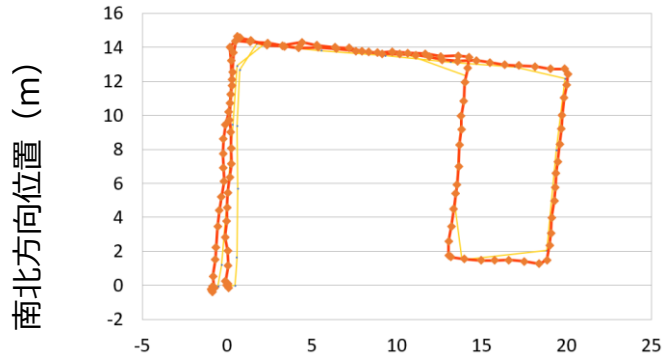
Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜No.3 3D都市モデル、サーバー処理

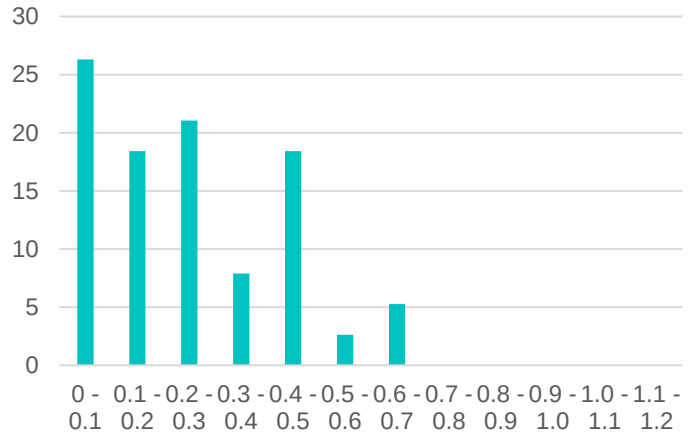
SLAMとRTKによる位置予測

SLAM:青
RTK：オレンジ

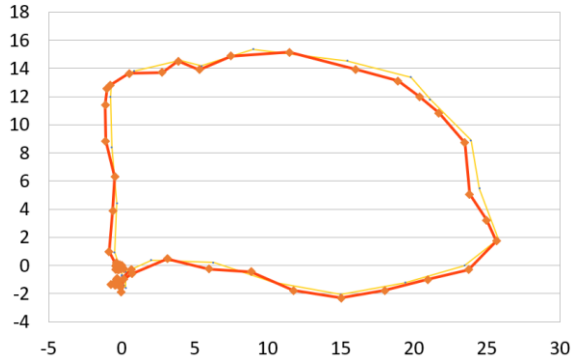
RTKとの差異の検出割合（%）

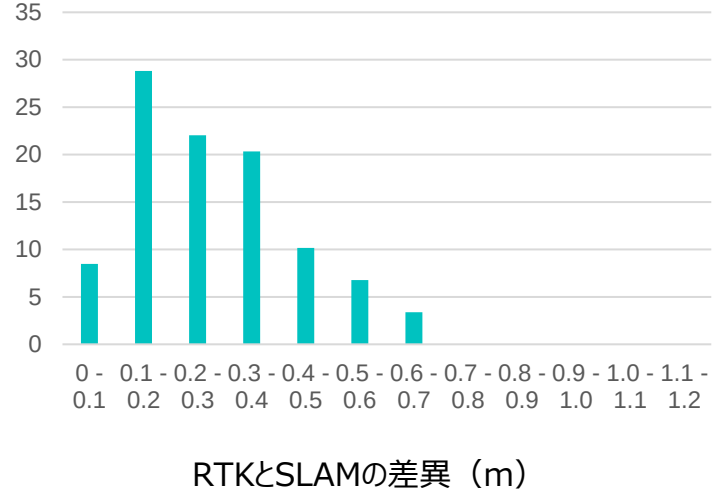
フライトA





フライトB





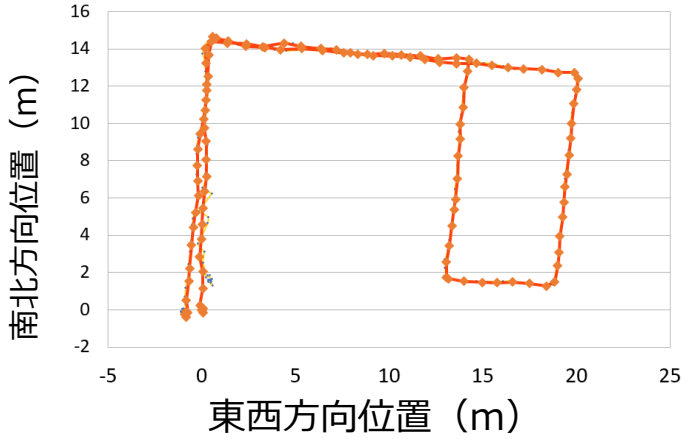
Ⅳ. 実証技術の検証 > 1. LiDAR SLAMによる自己位置推定の精度検証
②検証結果｜No.4 点群データ、サーバー処理

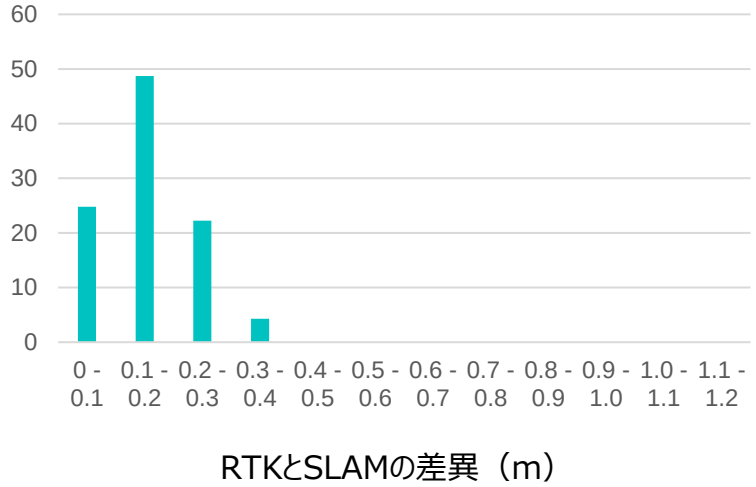
SLAMとRTKによる位置予測

SLAM:青
RTK：オレンジ

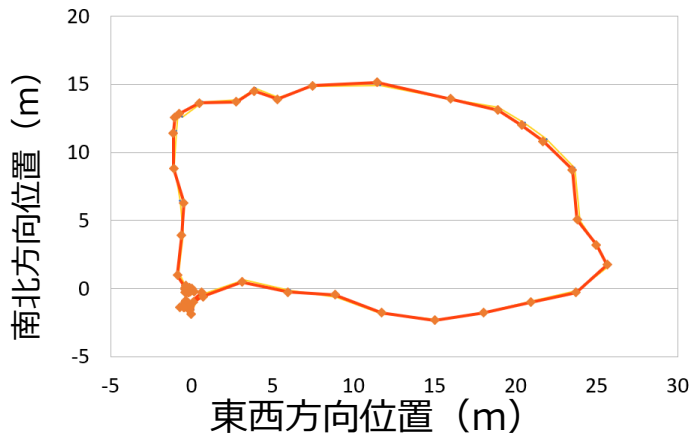
RTKとの差異の検出割合（%）

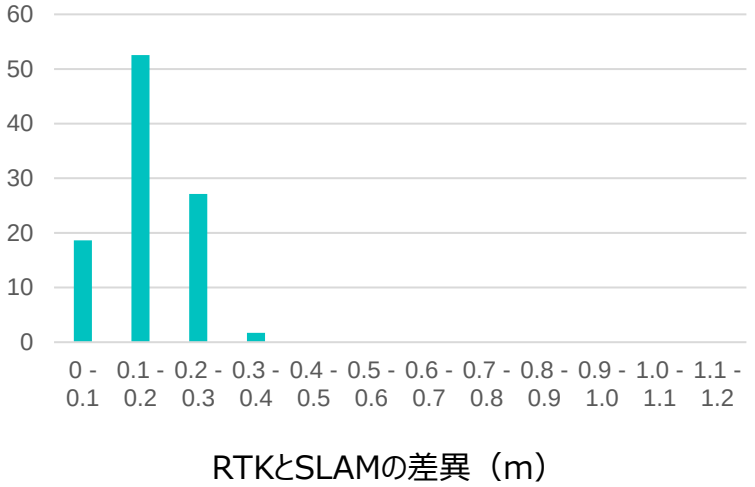
フライトA





フライトB





Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
①検証内容｜概要

目的	**Visual SLAM/3D都市モデル×エッジ処理×Shared Computingの検証** • Visual SLAM/3D都市モデル×エッジ処理とLiDAR SLAM/3D都市モデル×Shared Computingの構成パターンで、どの程度まで自己位置測定の精度を高く推定できるか検証する • Shared Computingを用いることやインターネット通信の速度が、どの程度ボトルネックとなるか確認する
実施期間	2023年2月27日
実施場所	山梨県甲府市
主な参加者	現場統括、パイロット、サブパイロット、ソフト・ハードエンジニア、現場進行：各1人
実施内容	• ステレオカメラおよび周辺の3D都市モデルをインストールしたコンピュータを搭載したドローンを、山梨県庁敷地内で飛行させ、Visual SLAMによる自己位置推定を行う • 自己位置推定の結果と、同時に計測を行うRTKを比較して自己位置推定精度を検証する • SLAMの演算処理を機体側で行った場合とサーバー側で行った場合を比較して、サーバー側で演算処理を行う場合のシステムの有用性を検証する • 自己位置推定精度検証およびサーバー演算の有用性検証から今後の課題を抽出する

①検証内容｜検証方法の詳細

Visual SLAMにおいて、2パターンの事前地図と4パターンの演算処理によるドローン飛行結果から、それぞれの自己位置推定精度を測定して比較検証する

検証内容

項目	内容
検証目的	Visual SLAMによる自己位置推定の精度検証
検証方法	• 精度が数cmレベルのRTKを真値と仮定して、4つのシナリオを使いVisual SLAMの位置精度を評価する（ログ分析） - 全球測位衛星システム（GPS）での誤差（～5m以内）を実用レベルの精度と仮定して、これを閾値に本SLAMシステムの有用性を評価する • マッチングするまでの時間を計測し、位置精度が担保されるまでの立ち上がり時間を評価する

検証シナリオ

シナリオ	事前地図	演算処理
No.1	3D都市モデル	エッジ（機体）
No.2	ステレオカメラによる事前取得画像データ	
No.3	3D都市モデル	シェアードコンピューティング（サーバー）
No.4	ステレオカメラによる事前取得画像データ	

①検証内容｜フライトルート

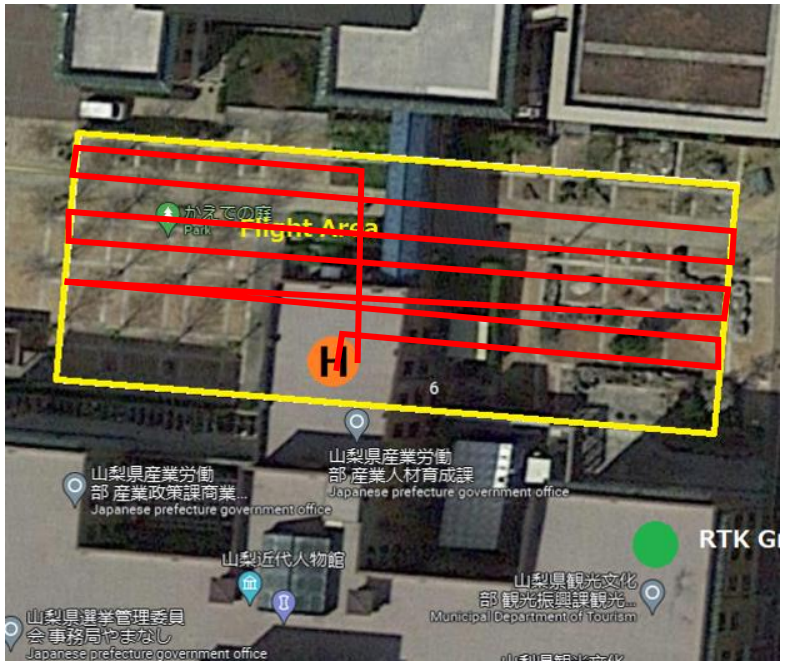
以下2つのフライトルートを設定し検証する

：検証エリア
：航行ルート

フライトA

- シンプルな長方形のルート
- 壁面に対して30m程度の距離の往復でどの程度の精度が出るのかを確認する

フライトB



- 壁面に沿って複数回往復するルート
- 10分程度の飛行において、安定した自己位置推定が可能なのかどうかを確認する

Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜サマリ

シナリオ	精度平均値（m）	標準偏差（m）	立ち上がり時間（sec）	示唆
No.1 3D都市モデル エッジ処理	フライトA：0.118 フライトB：0.271	フライトA：0.103 フライトB：0.311	12	3D都市モデルとVisual SLAMを組み合わせることで、一般的なGNSSを用いた場合と比較しても精度良く自己位置推定が可能であることが実証することができた
No.2 事前取得データ エッジ処理	フライトA：0.195 フライトB：0.180	フライトA：0.162 フライトB：0.140	14.3	事前取得したデータを用いた場合の精度は3D都市モデルを用いた場合と同等程度であった（3D都市モデルを用いた場合の数字は多少精度良く出ているが、標準偏差の値を考慮すると誤差範囲といえる）。このことから、Visual SLAMを用いようとしたとき、3D都市モデルを用いることで事前データの取得の工数を大幅に削減できる可能性が示された
No.3 3D都市モデル シェアードコンピューティング処理	フライトA：0.158 フライトB：0.498	フライトA：0.097 フライトB：0.302	17.3	SLAM処理をエッジで行った場合とリモートで行った場合での精度についての有意な差はみられなかった。このことから通信を用いることでの精度に対する大きな影響はなく、前述の回線速度の問題が改善されればクラウド環境で演算を行うというコンセプトは実現可能と考えられる
No.4 事前取得データ シェアードコンピューティング処理	フライトA：0.155 フライトB：0.249	フライトA：0.090 フライトB：0.166	12.8	No1とNo2の比較と同様に、リモート演算環境においても3D都市モデルを用いた場合と事前取得したデータを用いた場合の結果に大きな差異は見られなかった

Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜精度検証結果

シナリオ	(m)	フライトA				フライトB			
		最大	最小	平均	標準偏差	最大	最小	平均	標準偏差
No.1 3D都市モデル エッジ処理	合計	0.448	0.001	0.118	0.103	2.039	0.001	0.271	0.311
	1回目	0.344	0.005	0.111	0.081				
	2回目	0.448	0.001	0.121	0.113				
No.2 事前取得データ エッジ処理	合計	0.776	0.002	0.195	0.163	0.650	0.003	0.180	0.140
	1回目	0.349	0.005	0.138	0.090				
	2回目	0.776	0.002	0.225	0.183				
No.3 3D都市モデル シェアードコンピューティング処理	合計	0.397	0.004	0.158	0.097	1.960	0.001	0.498	0.302
	1回目	0.397	0.005	0.146	0.094				
	2回目	0.329	0.004	0.164	0.098				
No.4 事前取得データ シェアードコンピューティング処理	合計	0.406	0.005	0.155	0.090	0.999	0.010	0.249	0.166
	1回目	0.374	0.005	0.130	0.084				
	2回目	0.406	0.006	0.168	0.091				

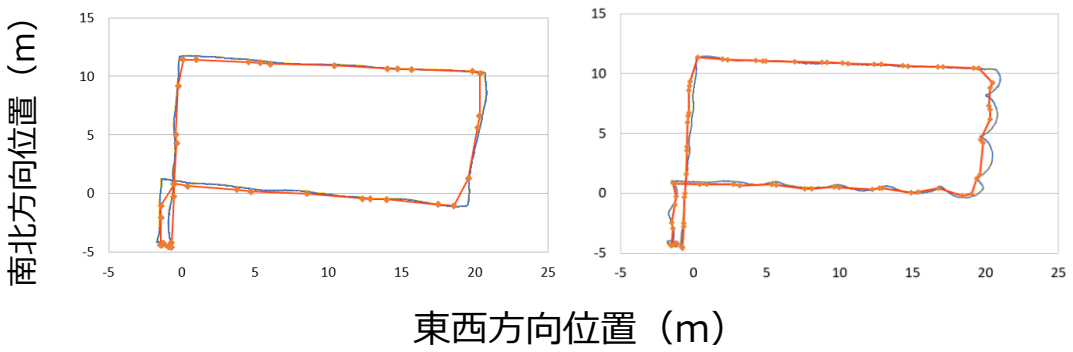
Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜No.1 3D都市モデル、エッジ処理

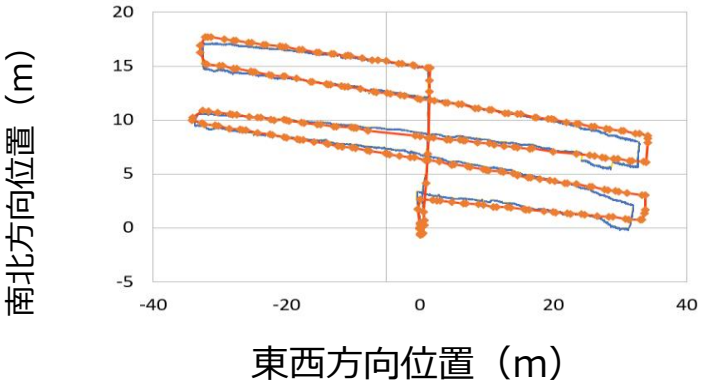
フライトA (左：一回目(1m/s)、右：2回目(0.5m/s))

フライトB

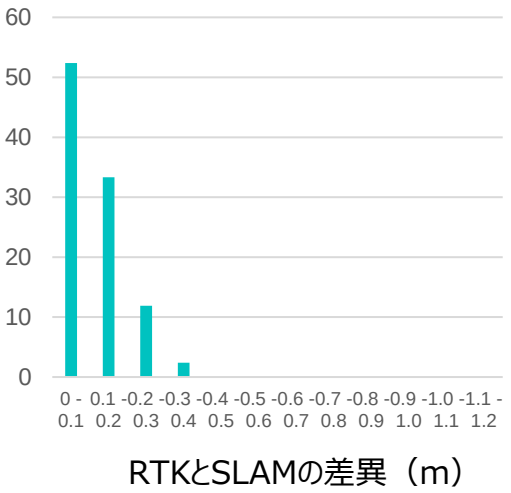
SLAMとRTKによる位置予測

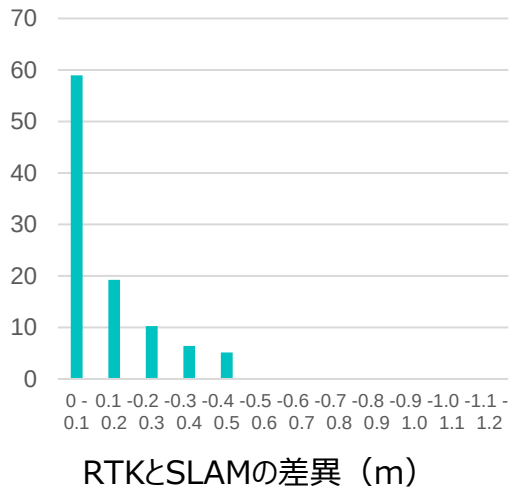
SLAM:青
RTK：オレンジ

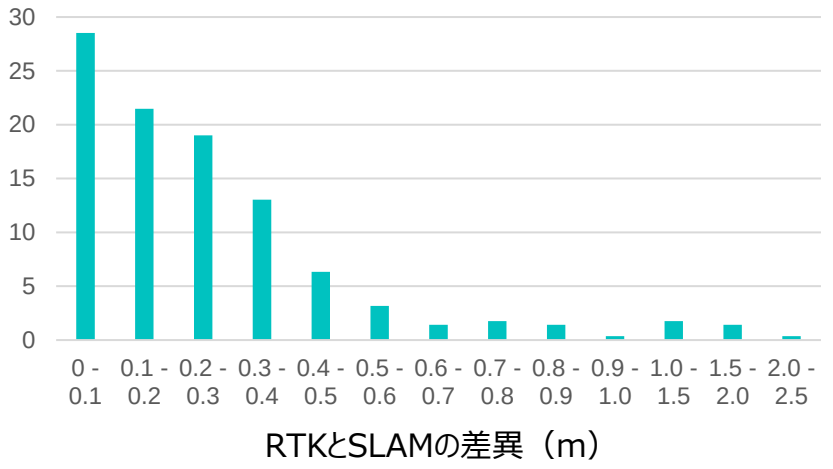




RTKとの差異の検出割合（%）







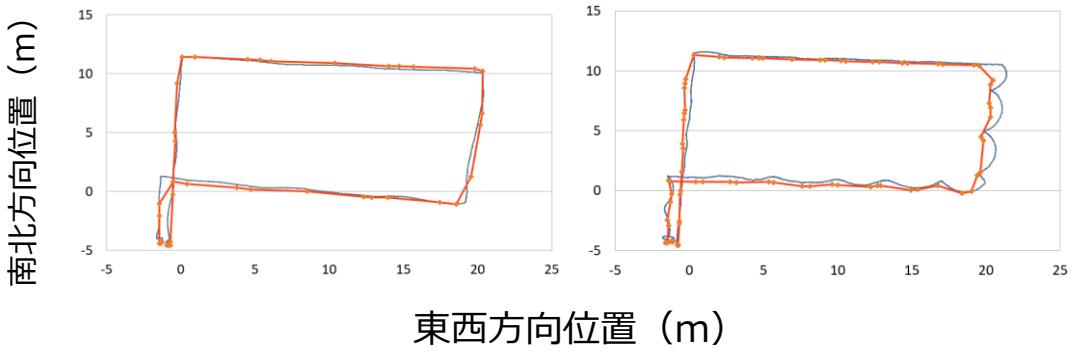
Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜No.2 事前取得データ、エッジ処理

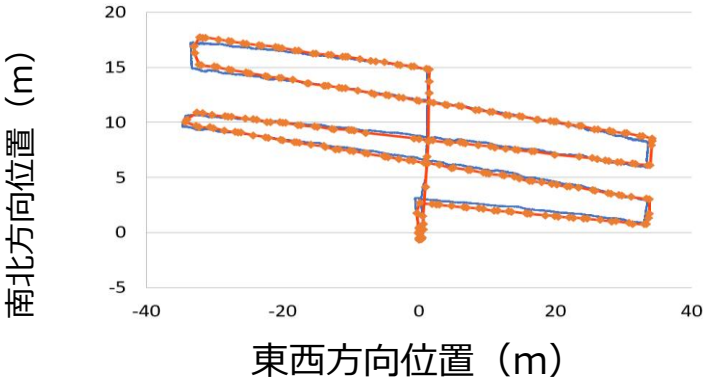
フライトA (左：一回目(1m/s)、右：2回目(0.5m/s))

フライトB

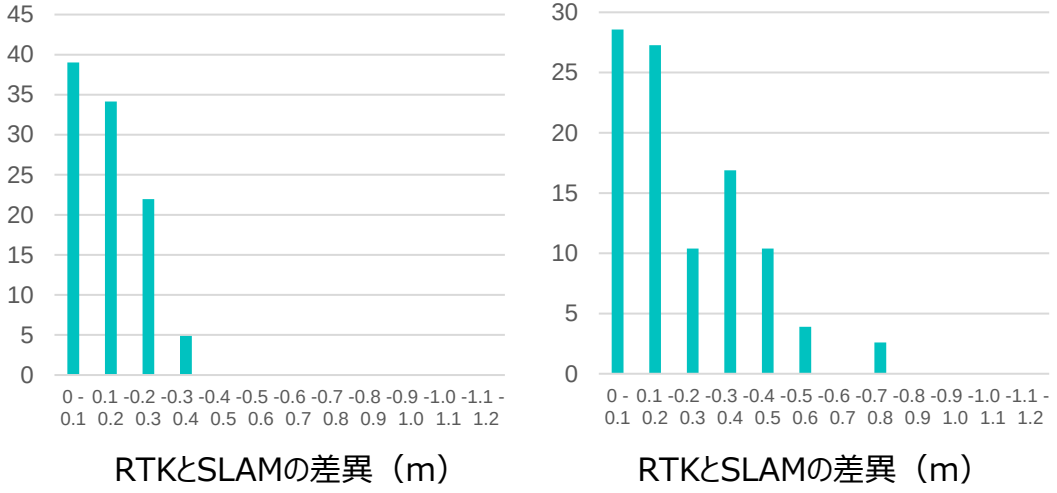
SLAMとRTKによる位置予測

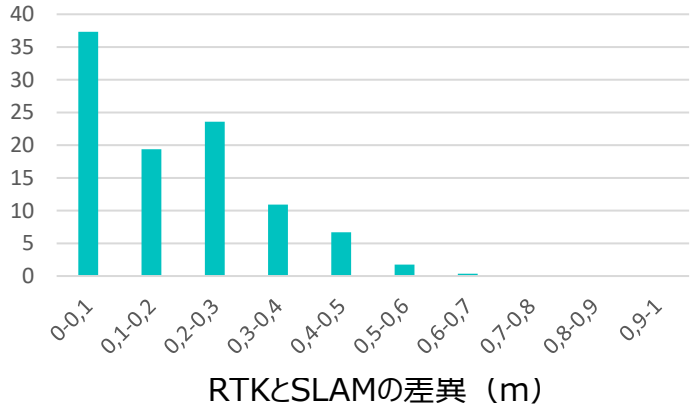
SLAM:青
RTK：オレンジ





RTKとの差異の検出割合（%）





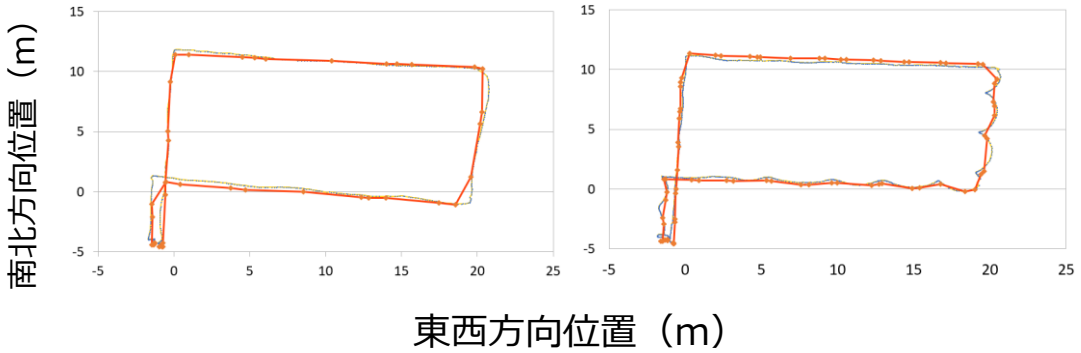
Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜No.3 3D都市モデル、サーバー処理

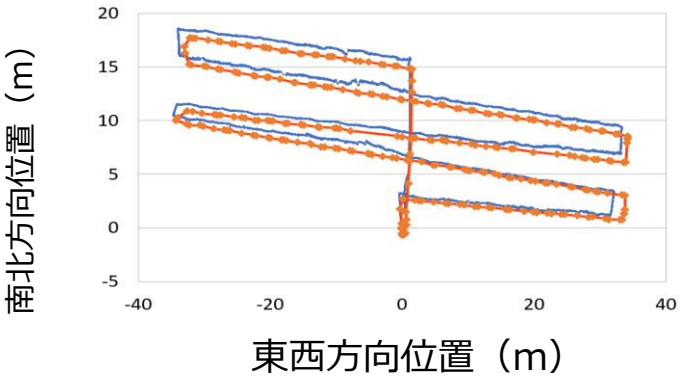
フライトA (左：一回目(1m/s)、右：2回目(0.5m/s))

フライトB

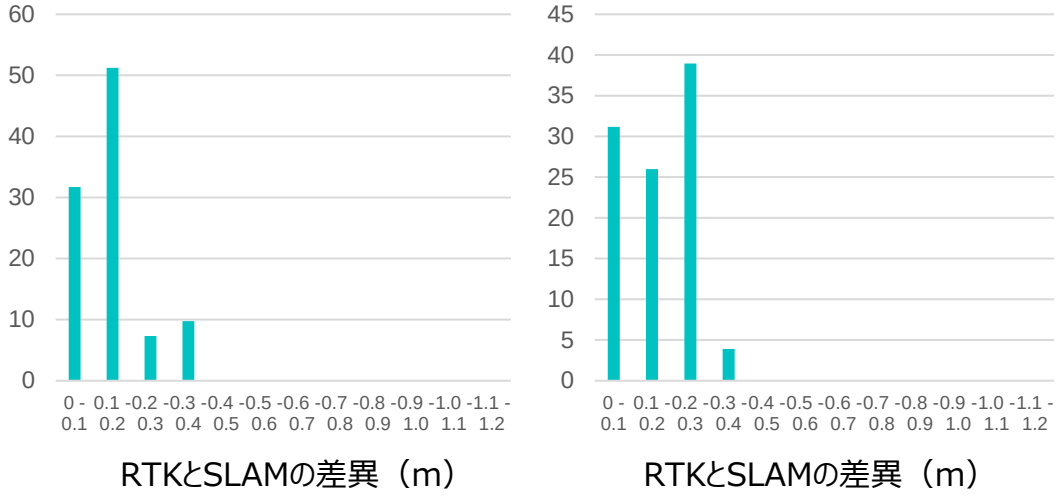
SLAMとRTKによる位置予測

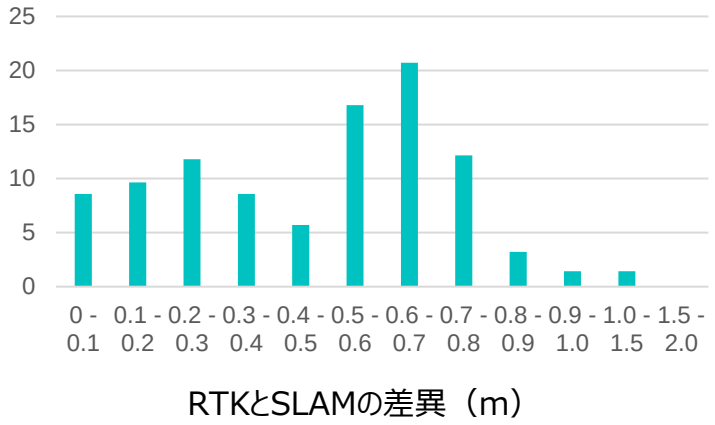
SLAM:青
RTK：オレンジ





RTKとの差異の検出割合（%）





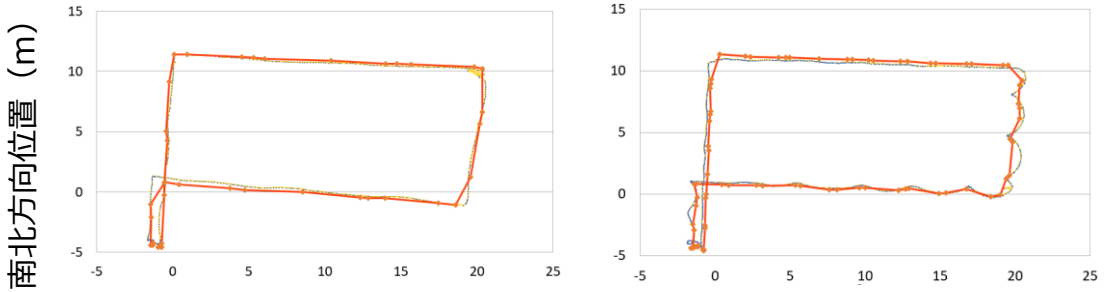
Ⅳ. 実証技術の検証 > 2. Visual SLAMによる自己位置推定の精度検証
②検証結果｜No.4　事前取得データ、サーバー処理

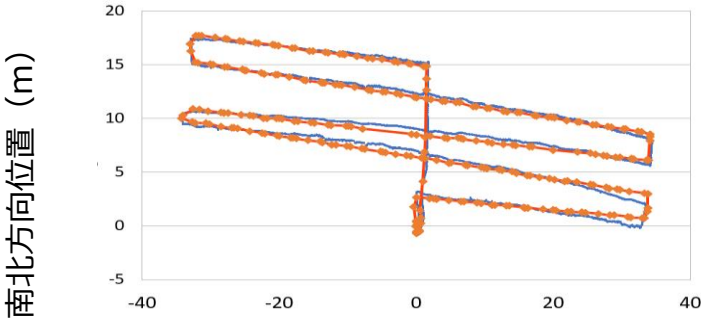
フライトA (左：一回目(1m/s)、右：2回目(0.5m/s))

フライトB

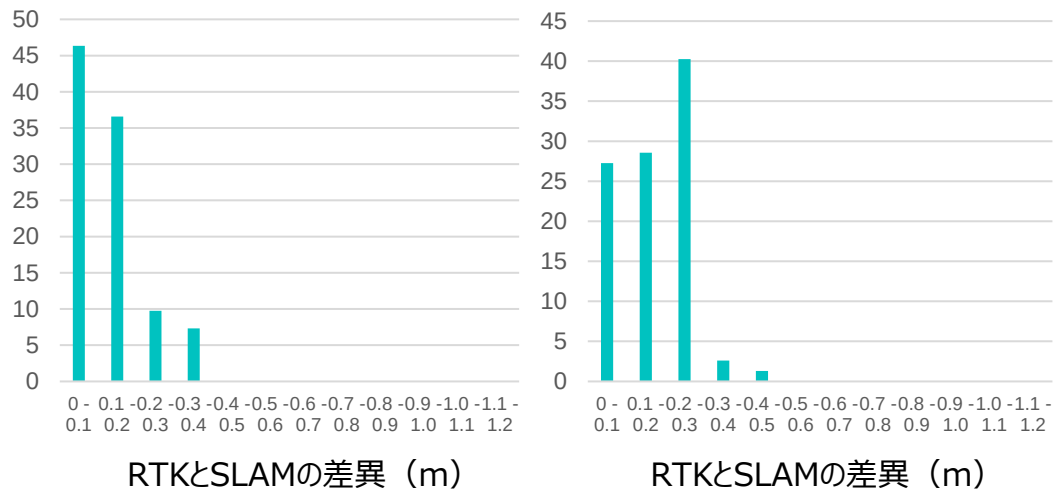
SLAMとRTKによる位置予測

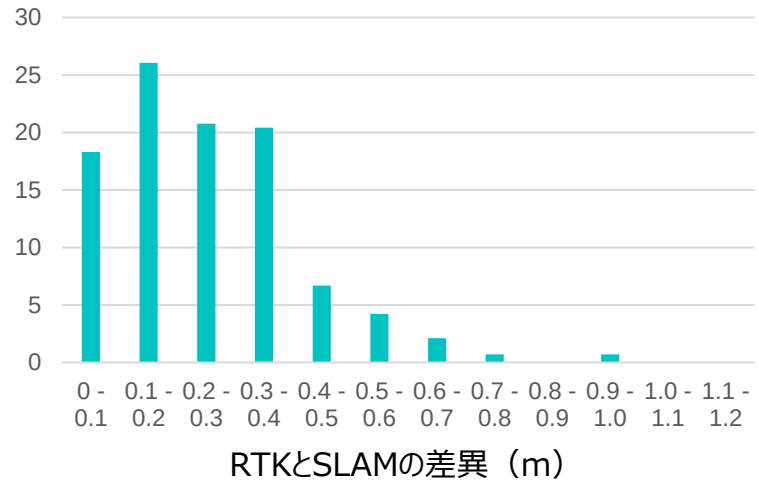
SLAM:青
RTK：オレンジ





RTKとの差異の検出割合（%）





②検証結果｜立ち上がり時間のサマリ

シナリオ	(sec)	フライトA	B	平均
No.1 3D都市モデル エッジ処理	平均	13	11	12
	1回目	8		
	2回目	18		
No.2 点群データ エッジ処理	平均	19.5	9	14.3
	1回目	18		
	2回目	21		
No.3 3D都市モデル シェアードコンピューティング処理	平均	18.5	16	17.3
	1回目	13		
	2回目	24		
No.4 点群データ シェアードコンピューティング処理	平均	17.5	8	12.75
	1回目	16		
	2回目	19		

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．実証システム

Ⅳ．実証技術の検証

Ⅴ．成果と課題

① 3D都市モデルによる技術面での優位性（1/2）

項目	想定される技術面での優位性
3D都市モデルの自己位置推定への活用	• 広域に整備され精度の高い3D都市モデルの存在は、今後都市部でドローンを飛行させるにあたり有用な事前地図として活用できるポテンシャルは十分あるといえる • 特に精度やデータ仕様、データ更新時期が公開されていることの意義は大きく、事前地図の精度や仕様が明確になっていることで、それを利用したSLAMの精度についてある程度見通しを立てることができる
LiDAR SLAM	• LiDAR SLAMの事前地図として3D都市モデルを活用することで、GNSSと比較しても高精度な自己位置推定が可能であるという結果が得られた。その精度は実際に現場を飛行させて得られたLiDAR点群をもとにした事前地図を使った場合の精度と比較しても遜色のないものであったことから、実際にLiDAR SLAMの運用を想定した場合、事前に地図取得のためのフライトを行うための工数を大幅に削減することができると考えられる • LiDAR SLAMの事前地図としては、LiDARセンサーによって取得したデータと事前地図を比較するため、建物の位置や高さだけではなく、できるだけ詳細な形状データがあることが望ましい • 本実証で飛行させた範囲においてはLOD2のレベルでも十分な精度を得ることができたが、より込み入った環境での活用においてはさらに詳細度の高いモデルが必要となる可能性もあり、計画的・継続的にデータ整備が進められているという点が3D都市モデルの技術的な優位性といえる

① 3D都市モデルによる技術面での優位性（2/2）

項目	想定される技術面での優位性
Visual SLAM	• 3D都市モデルを用いたシミュレーションを行うことで、Visual SLAMの自己位置推定の事前地図として用いることができることが確認された • あらかじめ飛行することなく事前地図を作成することでオペレーションの工数削減・安全性の向上という優位性がある • 自己位置推定の精度について、シミュレーションと実画像のマッチングにおける3D都市モデル側の形状精度（LOD）やテクスチャ解像度との関連性についてはさらなる検証が必要であるものの、今回のように必要に応じて同一モデルのテクスチャを張り替えて活用することができるなどの高いカスタマイズ性は3D都市モデルの活用という意味では大きな魅力である • 実際のVisual SLAMの運用を想定した場合、現在整備されている3D都市モデルの状況をPLATEAU VIEWを通してWebブラウザから簡易的なプレビューができることは有用である。Visual SLAMの事前地図として用いる場合、そのテクスチャの品質が重要となるが、利用可否チェックなどの上流段階でのフィジビリティ検証においてその確認を専門知識なく行えることで、スムーズなオペレーションが可能になると予想される
シェアードコンピューティング	• シェアードコンピューティングにおいても、3D都市モデルを使った自己位置推定が可能であった。エッジ処理ではデータ容量の制限から、飛行エリアを変えるたびに事前地図データの入れ替えが必要であるが、シェアードコンピューティングであれば、ストレージも大きくオンラインでのダウンロードも可能であることは優位性があるといえる • 本実証においては非常に限られた範囲の活用であったが、将来的に日本各地での活用を考えると、必要な地域の最新の3D都市モデルをシステム側でシームレスにダウンロードするなどの活用法が期待できる

② 3D都市モデルによるビジネス面での優位性

項目	想定されるビジネス面での優位性
公開性・拡張性	• 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、ドローン事業者としては追加コストや許可手続きを経ることなくスピード感をもって開発を進められることは大きな優位性といえる • ビジネスとしてのスケールを考えた場合、3D都市モデルの整備が進むことで特定の地域に限らない活用ができるようになるため、日本各地で飛行が求められるドローン役務との相性は非常によいと考える
信頼性	• 本実証実験で開発したシステムは安全な運航に関わるものであるため、その基礎データとなる3Dデータにも信頼性が求められる。その中で、開発者側、利用者側の双方にとって安全・安心な運航に寄与できるような、仕様や精度が明記され、データの信頼性・透明性の高い3D都市モデルは優位性がある

活用にあたっての課題

項目	活用にあたっての課題
【全般】 汎用的利用に向けた検証	• 本実証においては、システムの実現性の検証という観点から、LiDAR SLAM・Visual SLAMのどちらにおいても非常に限られた条件・環境における検証にとどまった • したがって、本開発で得られた知見の社会実装に向けては、様々な環境・条件において各システムが安定的に機能するのかを検証する必要があると考えられる
【LiDAR SLAM】 機器の最適化	• 本実証においては、機器のスペックが足りずにシステムが十分に機能しないというリスクを軽減するため、なるべく広い取得範囲で高い点密度が得られるLiDARを選定して実証を行った • 実際のドローンでの運用を考慮すると、なるべく軽量・小型のセンサーを用いることが望ましいため、得られる精度とセンサー仕様のバランスを考慮して最適化された機器選定やそのための検証が必要と考えられる
【LiDAR SLAM】 計算リソースについて	• LiDAR SLAMの自己位置推定について、GNSSと比較して十分な精度が得られたものの、計算負荷の都合で環境地図や演算する範囲のグリッドサイズを粗めに設定していた • 3D都市モデルの高精度な形状データをより活用するためには、より高い計算リソースを持つコンピュータを活用することが有効と考えられる
【Visual SLAM】 カメラの画角	• 本実証においては、常に機首方向を壁面に向けて同じ面の画像を取得し続けることでVisual SLAMにおける自己位置推定を行った • 実際の活用場面においては、常に建物の方向を向き続けるという飛行方法は現実的ではないため、カメラを複数の方向に取り付ける等より汎用的な利用に向けての開発・検証が必要と考えられる
【シェアードコンピューティング】 携帯通信網	• 本実証においては、携帯電話通信網の速度がボトルネックとなり、クラウド上でデータ処理を行うという当初の想定で実証を行うことができなかった • 通信インフラの改良は日進月歩で進んではいるものの、SLAM処理のためのクライドコンピューティングに向けては通信網の改善が社会的な課題となる

用語集（1/5）

用語		内容
ア行	RTK - GNSS	• Real Time Kinematicsの略 • 一般的なGNSS情報に加え、基準局を設けてその地点における位置情報を元に補正データを演算し、GNSS単独での測位値を補正して高精度な位置推定を行う手法 • 一般的にcm単位の位置情報精度が得られると言われている
	Rviz	• ROSで用いることができる3D可視化ツール
	IMU	• 「Inertial measurement unit」。慣性計測装置 • 一般的には3軸のジャイロセンサーと3方向の加速度計を用いて、角速度と加速度を計測する装置
	エッジ処理	• ドローン搭載の端末（コンパニオンコンピュータ）上でなされる演算
	LTE	• 「Long Term Evolution」の略称。携帯電話などに用いられる通信規格 • 当初は3Gと4Gの中継ぎとしての役割であったが、LTEを4Gと呼ぶことも認められるため、4Gの通信規格を指す場合もある
	オドメトリ	• 自己位置推定手法の一つ • 初期位置・姿勢から現在までの移動量の積み重ね（積分）で現在位置を推定する手法 • カメラ（モノラル・ステレオ）の取得画像から画像処理により微分量を推測する手法を特にビジュアル・オドメトリと言う
	Octomap	• 八分木データ構造を採用した3D占有グリッドマップ構造 • 利用時の柔軟性と容量効率に優れる

用語集（2/5）

用語		内容
カ行	カルマンフィルタ	• 誤差を含む観測値から時系列データを推定するために用いられるフィルタ
	Kd Tree	• 「k dimension tree（k次元ツリー）」の略称 • k次元のデータを分類する空間分割データ構造 • 最近傍探索を行うのに適したデータ構造であり、各次元において中間的な数字（中央値を採用する場合が多い）で分割し、その分割領域までの距離をもとに探索有無の判断を行うことで、効率的な近傍探索が可能となっている
	COSMOS	• A.L.I. Technologiesが自社で開発する運航管制アプリケーション「Centralized Operating System for Managing Open Sky」の略称
	コンパニオンコンピュータ	• 構成上はフライトコントローラの上位に位置し、フライトコントローラの情報を抽出したり、センサーのデータを集約して演算を行ってり、フライトコントローラや外部サーバーにコマンドを出力する制御装置
サ行	GNSS	• 「Global Navigation Satellite System：全球測位衛星システム」の略称 • 米国のGPS、日本の準天頂衛星（QZSS）、ロシアのGLONASS、欧州連合のGalileo等の総称 • ドローンの屋外でのナビゲーションで一般的に用いられる
	Shared Computing	• 高性能なGPUを複数搭載したクラスターマシンを用いて、その演算リソースをシェアすることで効率的に稼働するという概念
	SLAM	• 「Simultaneous Localization And Mapping」 自己位置推定と環境地図作成を同時に行う技術。LiDARやカメラ（モノラル、ステレオ）などを使う方法が主流 • 出力は主に環境地図情報（点群データなど）および自己位置・自己姿勢情報（変位・向き）

用語集（3/5）

用語		内容
サ行	Cesium	• 米国のCesium GS, Inc.が提供する3D地図プラットフォーム。主にWebブラウザ上で用いられる3D地図として活用される
タ行	ToFセンサー	• 「Time of Flightセンサー」の略称 • センサーから発せられたパルス信号からセンサー内の受光素子に戻ってくるまでの時間を計測して距離を測るセンサー
	特徴点抽出	• 画像の中から特徴的なポイント（主に角など）を抽出するアルゴリズム • AIによる画像解析などの用途に用いられる
	特徴量	• 抽出された特徴点について、その特徴を記述するパラメータ • 特徴点の分類や照合などに用いられる
	ドローンナビゲーションシステム	• ドローンのナビゲーションを支援または自動化するために設計されたハードウェアおよびソフトウェアからなるシステ • ドローン機体の位置や指定箇所への飛行ルートを決定する機能を持つ
ナ行	Node	• ROSにおける特定の機能を持ったプロセス単位 • ROSの構成としては、このノードを適切に構成し、それらをトピックやサービスと呼ばれる方法でデータ通信を行い、ロボットのアプリケーションを作成する
ハ行	Bag-of-Words	• 自然言語処理において、文章中の単語の出現回数をもとに評価・分類する手法 • 出現する順番は考慮しないのが特徴である • この評価分類法を画像分類の応用したものをBag-of-visual-Wordsと呼ぶ

用語集（4/5）

用語		内容
ハ行	Pixhawk	• ドローンなどに用いられれるオープンソースハードウェアのフライトコントローラ
	フライトコントローラ	• ドローンの制御の中心となる制御基板 • センサーなどのデータ処理やそのデータをもとにした姿勢制御等などドローンの基本的な制御を行う
マ行	MAVLink	• ドローン用に設計された非常に軽量で汎用的な通信プロトコル
ヤ行	Unity	• リアルタイム開発プラットフォーム • ゲームやシミュレータなど3D空間を活用したソフトウェアの開発などに多く用いられる
	UTM	• 「UAS Traffic Management：ドローン運航管理システム」の略称 • ドローンの運航が複数ある空域においても、目視外環境での安全かつ効率的な運航を実現するための概念 • 主要なアーキテクチャではFISS（Flight Information Management System：運航管理統合機能）、SDSP（Supplemental Data Service Proveider：情報提供プロバイダ）、UASSP（UAS Service Supplier：運航管理サプライア）、運航管理オペレータ、USAから構成される
ラ行	LiDAR	• 「Light Detection and Ranging、Laser Imaging Detection and Ranging」の略称 • 光を用いたリモートセンシング技術で、照射した光の角度と反射までの時間をもとに周辺環境をセンシングする • またはそれに使うレーザー装置そのものをLiDARと呼ぶ
	粒子フィルタ	• 特定のパラメータの値を知りたい場合、多数の粒子を生成し、各粒子その尤度を演算したうえで重み付き平均としてそのパラメータの推定するアルゴリズム • 汎用性が高いが、多数の粒子に対して計算を行うことから高い計算リソースが求められる

用語集（5/5）

用語		内容
ラ行	レベル4飛行（ドローン）	• 有人地帯（第三者上空）での補助者なし目視外飛行を指す • 2022年10月の航空法改正により解禁された
	ROS	• 「Robot Operating System」の略称 • ロボットのアプリケーション作成を支援するオープンソースのソフトウェアライブラリおよびツール群 • Operation Systemの名前を冠しているが、厳密にはオペレーションシステムではなく、オペレーションシステムとアプリケーションの中間に位置するミドルウェアの位置づけである

ドローンリアルタイム・ナビゲーションシステム
技術検証レポート

令和5年3月　発行
委託者：国土交通省 都市局 都市政策課
受託者：株式会社 A.L.I. Technologies

本報告書は、株式会社 A.L.I. Technologiesが国土交通省との間で締結した業務委託契約書に基づき作成したものです。受託者の作業は、本報告書に記載された特定の手続や分析に限定されており、令和5年3月までに入手した情報にのみ基づいて実施しております。従って、令和5年4月以降に環境や状況の変化があったとしても、本報告書に記載されている内容には反映されておりません。