車両の走行データとレーンレベル道路地図を用いた点群データの検索システムの実装

井川　元[1,a)]　渡辺　陽介[2]　高田　広章[1,2]

概要：近年，自動運転に関する研究が進められている．自動運転や運転支援システムが安全に運用されるためには，誤差の少ない自己位置推定が必要と考えられている．自己位置推定には，3 次元の点の集まりである点群データを利用する．名古屋大学発となる自動運転の実験システム Autoware では，事前に収集した点群データと，センサーによって動的に測量したデータを用いて自己位置推定が行われている．点群データはファイルによって管理されており，それを一括で読み込んで利用している．しかし，点群データは膨大なため，大領域で実験を行うときには，データサイズの問題からすべての点群をメモリに読み込むことはできない．したがって，自己位置推定のためには，必要な点群のみを読み込んで利用することが求められる．そこで，本研究では点群データ及びレーンレベル道路地図を RDBMS（PostgreSQL，PostGIS）に格納し，また，動的に得られる車両の走行データ，レーンレベル道路地図を利用することで，必要な点群のみを検索によって取得する方法を検討，実装してメモリ使用量および検索時間の評価を行った．その結果，点群データを一括で読み込んだ場合よりも本研究で実装した方法のほうがメモリ消費量の観点から優れていることが分かった．

キーワード：点群データ，高精度地図，自己位置推定

1. はじめに

近年，自動運転に関する研究が進められている [1]．自動運転には，誤差の少ない自己位置推定や周辺環境の認知が必要と考えられている．自己位置推定を行う手法の一つに，NDT スキャンマッチング [2], [3] がある．NDT スキャンマッチングには 3 次元の点の集合である点群データを利用する．点群データは MMS（モービルマッピングシステム）[4] によって収集，作成される．名古屋大学内の点群データを**図 1** に示す．NDT スキャンマッチングは，事前に作成した点群データとセンサによって動的に測量した点群データを利用して行われる．

名古屋大学発自動運転の実験システム Autoware[5] は，Linux と ROS[6] をベースとしたシステムで，レーザやカメラ，GNSS などのセンサを利用して自己位置推定や地図生成，物体認識などを行う機能をもっている．本研究で着目するのは，自己位置推定である．Autoware では自己位置推定に先に述べた NDT スキャンマッチングを用いることができる．このとき，事前に作成した点群データは pcd ファイルに保存されており，実験開始時に点群データを一括でメモリに読み込んでいる．

しかし，点群データは膨大である．データ点数および量は名古屋大学内だけでも約 7,000 万点，10GB に達する．より広範囲の領域の全点群データをメモリに同時に読み込むことは現実的でない．そのため，自己位置推定に必要となる点群データのみを読み込むことが必要になると考えられる．本研究では，点群データを RDBMS（PostgreSQL，PostGIS）によって管理し，さらに，車両の走行データとレーンレベル道路地図を利用することで「自車両のいるレーン周辺」といった自己位置推定に必要となる点群データのみを検索するシステムの検討，実装を目的とする．

点群データを，レーンレベル道路地図に含まれるレーンの情報と関連付けて効率よく取得するための工夫として，以前 [7] で提案したテーブル構成方法を利用している．[7] の実験では，RDBMS 単体に対して 1 回のクエリ結果を取得するまでの検索時間の測定しか行われていなかったが，本研究では，取得した点群データを実際に自己位置推定の処理へと送るとともに，車両の移動に合わせて連続的に点群データを検索し続ける実験を行っている．

本稿の以降の構成は以下のとおりである．第 2 節では，関連研究について紹介する．第 3 節では，Autoware にお

[1] 名古屋大学大学院情報学研究科
[2] 名古屋大学未来社会創造機構
[a)] gengen@ertl.jp

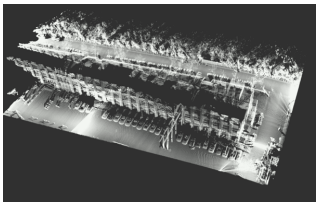

図 1　点群データ

図 2　実験車両

ける自己位置推定の流れを示す．第 4 節では，道路地図について述べる．第 5 節では，本研究で実装した検索システムについて述べる．第 6 節では，検索システムの実験結果について述べる．最後に第 7 節では，本研究のまとめと今後の課題について述べる．

2. 関連研究

点群データの管理方法についての先行研究が存在する．Cura ら [8] は，点群サーバシステムを提案している．RDBMS として PostgreSQL を利用しており，点群データの管理に OSS である pgPointCloud[9] を利用している．これにより，点群データを点ごとではなく点のグループ（Patch）で管理し，Patch に索引をつけている．しかし，高精度地図と点群データの関係性を考慮して検索の効率化を図っていないという点が本研究と異なる．

3. Autoware

自動運転車には，**図 2** に示すように GNSS や LiDAR などの多くのセンサが搭載されることが想定される．GNSS では，単体で大まかな自己位置推定を行うことができるが，精度が不十分であったりトンネル内など測位ができない状況が発生してしまう．しかし，LiDAR は周囲の点群データを取得して，事前に収集した点群データと比較することで GNSS よりも高精度な自己位置推定を行うことができる．

本研究のシステムの対象とした Autoware の概要と自己位置推定の流れについて述べる．Autoware[5] は Linux と ROS[6] をベースとした自動運転システム用オープンソースソフトウェアであり，プロセス間の通信に ROS を利用している．ROS は publish/subscribe 通信方式を提供している．ROS では，プロセスのことをノードといい，publish/subscribe 通信とはノードがデータをトピックに送信（publish），またはデータをトピックから受信する（subscribe）ことでノード間の通信を行うモデルである．各ノードはトピックを介することで通信相手となるノードを意識することなく非同期通信を行うことができる．データをトピックに送信するノードを publisher，トピックから受信するノードを subscriber と呼ぶ．通信モデルを**図 3** に示す．また，ROS には，Publish されたトピックを rosbag として保存しておき，後に再生する機能がある．

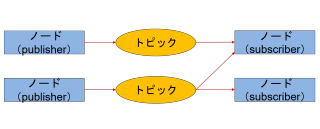

図 3　publish/subscribe 通信

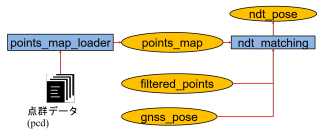

図 4　Autoware による自己位置推定

Autoware における自己位置推定にも publish/subscribe 通信が利用されている．それを**図 4** に示す．ノード points_map_loader で，pcd ファイルに保存されている点群データを points_map として publish する．そしてノード ndt_matching は，走行中に得られた gnss_pose（GNSS）や filtered_points（走行中に得られた点群データをダウンサンプリングしたもの），points_map を subscribe して，NDT スキャンマッチングにより自己位置推定を行う．そして自己位置推定によって得られて結果を ndt_pose として publish する．本研究では，図 4 の points_map_loader を置き替えている．

4. 道路地図

自動運転にむけて，ダイナミックマップ [10], [11] の研究が進められている．ダイナミックマップは，静的情報である高精度地図をベースに渋滞情報や周辺車両の情報などの動的情報を組み合わせた地図である．

高精度地図の形式として，文献 [12] では，用途に合わせた以下に示す詳細度の異なる 3 レベルの道路情報を提案している．

- リンクレベル：グラフデータとして表現され，各交差点をノード，交差点間を結ぶ道路を一本のリンクとして表現する．
- レーンレベル：グラフデータで表現され，各道路のレーン一本一本について区別された詳細度を持つ道路情報である．
- 道路形状レベル：カメラやレーダ，LiDAR によって観測した生データ相当の情報である．

ここで，自己位置推定に必要となる点群データは，道路形状レベルに該当している．しかし，異なる詳細度の道路情報を組み合わせた検索について言及されていないが，実際にはレーンレベルの道路情報によって今いるレーンの先の ID を取得し，その周囲の道路形状レベルの道路情報を取得する，といったように組み合わせた検索が考えられる．

そこで，本研究では，あらかじめレーンレベル道路地図と点群データの対応付けを行い，車両の走行データとレーンレベル道路地図から，今いるレーン情報を取得した後，点群データを検索するシステムの検討，実装を行っている．

5. 検索システム

自己位置推定のために用いられる点群データを走行データとレーンレベル道路地図を利用して検索するシステムの

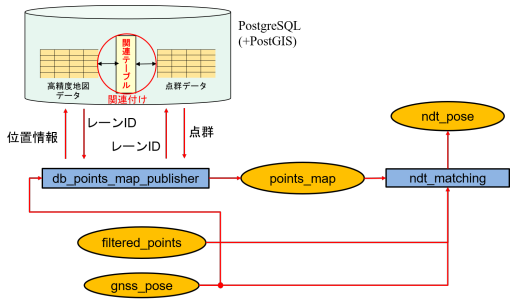

図 5　点群検索システムの概要

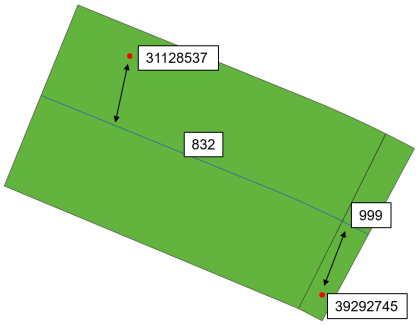

図 6　関連付け手法（レーン左右重視型）

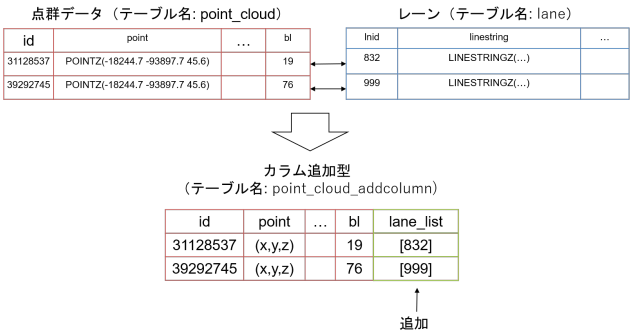

図 7　関連テーブルの作成（カラム追加型）

実装を行った．システムの概要を**図 5** に示す．システムの入力と出力は以下の通りである．

入力：　GNSS からの荒い自己位置情報

出力：　自車両のいるレーンと，その先のレーンの点群

また，システムが入力を受け取ってから出力を返すまでに行う動きを以下に示す．

(1) GNSS で得られた位置情報から最近傍のレーンを求める．

(2) そのレーン（と n ホップ先のレーン）周辺の点群データを取得する．

(3) トピック points_map のデータ型に変換して Publish する．

レーン周辺の点群データを検索する際には，筆者が提案した「関連付け」を用いている（詳細は [7] を参照）．関連付けは，あらかじめ高精度地図と点群データを「関連付け手法」によって，対応付けを行い，それを「関連テーブル」

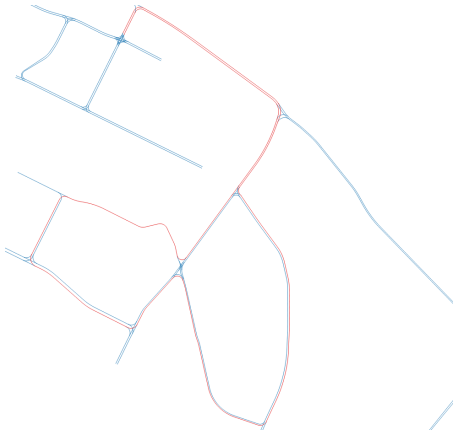

図 8　走行したレーン

に記録しておくことで空間演算を行うことなく，点群データの検索を行う．本研究では，関連付け手法としてレーンを基準に左右方向へ直角に 50m 広げてできる領域内の点群データの ID とそのレーン ID の対応付けを行うレーン左右重視型を，関連テーブルとして点群データのテーブルに対応づけられたレーン ID をもつカラムを追加するカラム追加型を適用している．レーンを指定することで，その周囲の点群データを空間演算を行わず，取得することを可能にしている．それぞれの例を**図 6**，**図 7** に示す．

6.　実験

本研究で実装した点群データの検索システムを用いて，点群データを Publish する実験を行った．実験で使用した点群データは事前に MMS によって収集されたもので，総数は約 7,000 万点である．レーンレベル道路地図は名古屋大学のもので，レーンは分岐，合流点ごとに分割されている．また，実験の入力となった走行データは，事前に GNSS を搭載した自動車を走行させて得られた実データを記録したものを再生して与えた．実験で使用した走行データは，名古屋大学内の**図 8** の赤線部分を走行したときに収集したデータである．実験を行った PC の仕様を**表 1** に，PostgreSQL で変更した設定を**表 2** に示す．表 2 の shared_buffer と work_mem は，それぞれ 1,2,3 で設定を変更して実験を行った．

全点群データを読み込んだ時と，提案手法を PostgreSQL の設定を変更して使った時のメモリ使用量の変化を**図 9** に示す．グラフの横軸は経過時間，縦軸はメモリの初期値からの差を表している．なお，実験時には現在いるレーンから 2 ホップ先までのレーン周辺の点群データを Publish している．

図 9 から，全点群データを読み込むよりも本研究で実装した検索システムの方がメモリの使用量が小さいことが確認できた．ただし，実際に Publish した点群データの量よりもメモリの使用量は増加している．これは，PostgreSQL や OS のキャッシュによって，Publish している点群デー

表 1　PC の仕様

OS	Ubuntu 16.04.5 LTS
CPU	Intel Xeon W-2133（6 コア，3.6GHz）
DB version	PostgreSQL 10.6，PostGIS 2.5
ディスク	SSD 512GB（SATA 6Gb/s）
メモリ	32GB DDR4 SDRAM（8GB × 4）

表 2　変更した PostgreSQL の設定

	設定 1	設定 2	設定 3
shared_buffer	4GB	2GB	1GB
work_mem	1GB	512MB	256MB
maintenance_work_mem	256GB	256GB	256GB
effective_cache_size	16GB	16GB	16GB

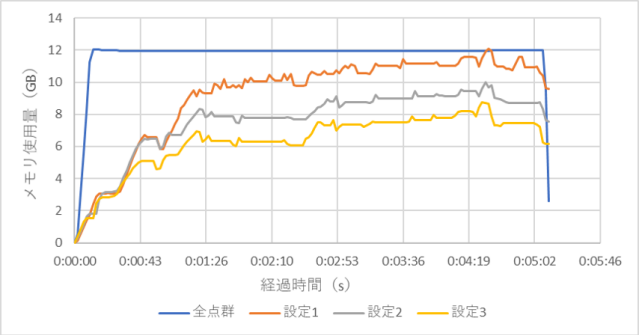

図 9　メモリ使用量

表 3　点群データの検索および変換時間

取得した点群データ数	検索時間（s）	変換時間（s）
432728	0.227	0.09
684455	0.459	0.133
1405289	1.072	0.26
2673536	2.128	0.488

タ以外もメモリに読み込まれているからだと考えられる．PostgreSQL の shared_buffer，work_mem の値を小さくすることで，PostgreSQL の使用できるキャッシュ量は小さくなることと，グラフの設定 1，設定 2，設定 3 の順にメモリ使用量が小さくなっていることからもキャッシュによるメモリ使用量が大きいことが分かる．したがって，PostgreSQL に割り当てるキャッシュ量を小さくすれば，メモリの使用量を小さくすることができる．ただし，その場合，データを外部ストレージから取り出すことになる可能性が高くなるため，平均検索時間は長くなってしまうと考えられる．

次に，点群データの検索時間と，取得した点群データを points_map のデータ型に変換する時間の測定を行った．結果を**表 3** に示す．なお，点群データの検索はデータがキャッシュにのっている状態で行っている．この結果から，点群データの検索および変換時間は点群データ数にほぼ比例することが分かる．なお，検索は車両のいるレーンが切り替わるタイミングで行う．次のレーンへ移動する前に，先の点群データが取得できていればよいので，検索時間が 2 秒でも実用性はあると考えられる．

7. まとめと今後の課題

本研究では，自己位置推定の手法の一つである NDT スキャンマッチングで使われる点群データを車両の走行データとレーンレベル道路地図を用いて検索するシステムの検討，実装を行った．

現在，検索を行うとき前回検索した点群データについて考慮していないことから，すでに取得した点群データも重複して検索している．しかし，検索時間を小さくするために，すでに取得した点群データを検索しないよう前回の検索結果との差分のみ点群データを取得することが今後の課題として挙げられる．さらに今回の実験では，2 ホップ先までのレーンの点群データを取得していたが，レーンによって長さは異なるため，ホップではなくレーンの長さでどこまで先読みを行うかを決定する必要があると考えられる．

謝辞　本研究は，JST，OPERA，JPMJOP1612 の支援を受けたものである．

参考文献

[1] 青木 啓二：自動運転車の開発動向と技術課題：2020 年の自動化実現を目指して，情報管理，Vol. 60, No. 4, pp. 229–239 (2017).
[2] Peter Biber, Wolfgang Strasser : The Normal Distributions Transform: A New Approach to Laser Scan Matching, *IEEE International Conference on Intelligent Robots and Systems*, Vol. 3, pp. 2743–2748 (2013).
[3] Eijiro Takeuchi, Takashi Tsubouchi : A 3-D scan matching using improved 3-D normal distributions transform for mobile robotic mapping, *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 3068–3073 (2006).
[4] アイサンテクノロジー（株）：MMS（モービルマッピングシステム），`http://www.whatmms.com/`.
[5] Autoware: Autoware.AI, `https://www.autoware.ai/`.
[6] ros.org: ROS Wiki, `http://wiki.ros.org`.
[7] 井川 元ら：高精度地図データおよび点群データの検索効率化手法，第 11 回データ工学と情報マネジメントに関するフォーラム (DEIM2019)，F5-4 (2019).
[8] RémiCura, Perret, J., Paparoditis, N. : A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing, *ISPRS Journal of Photogrammetry and Remote Sensing*, Vol. 127, pp. 39 – 56 (2017).
[9] Ramsei, P.: A PostgreSQL extension for storing point cloud (LIDAR) data, `https://github.com/pgpointcloud/pointcloud`.
[10] 小山浩ら：自動走行におけるダイナミックマップ整備，「システム/制御/情報」，Vol. 60, No. 11, pp. 463–468 (2016).
[11] 渡辺 陽介ら：協調型運転支援のための交通社会ダイナミックマップの提案，第 7 回データ工学と情報マネジメントに関するフォーラム (DEIM2015)，F6-6 (2015).
[12] ダイナミックマップ 2.0 コンソーシアム：名古屋大学 COI 高精度地図フォーマット仕様書，`http://www.nces.i.nagoya-u.ac.jp/dm2/COImap_20170906.pdf`.