DEIM Forum 2019 H5-4

高精度地図データおよび点群データの検索効率化手法

井川　元† 渡辺　陽介†† 高田　広章††,†††

† 名古屋大学工学部　〒 464–8601 愛知県名古屋市千種区不老町
†† 名古屋大学未来社会創造機構　〒 464–8601 愛知県名古屋市千種区不老町
††† 名古屋大学大学院情報学研究科　〒 464–8601 愛知県名古屋市千種区不老町
E-mail: †,†††{gengen,hiro}@ertl.jp, ††watanabe@coi.nagoya-u.ac.jp

あらまし　将来に向けて，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，ダイナミックマップが必要性を増してきている．ダイナミックマップとは，従来のナビゲーション用の地図よりも細かい粒度（レーン単位など）で記述された静的な高精度地図に，渋滞情報や事故による通行規制などの動的な位置情報を組み合わせたものである．また，自動運転システムでは事前に取得した点群データと動的に計測した点群データを照合して自己位置を推定している．名古屋大学発オープンソースの自動運転実験システムでは，自己位置推定の時，点群データを最初に一括で読み込みメモリ上に展開している．しかし，点群データは膨大なため DBMS によって管理し，必要な点群を適宜検索によって取得するべきだと考えられる．本研究では，高精度地図データと点群データを RDBMS（PostgreSQL，PostGIS）に取り込み，あらかじめ両者を関連付けておくことで点群の検索効率を向上させる手法を検討し，その性能について実験を行った．

キーワード　点群データ，高精度地図，空間検索，ダイナミックマップ

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]．自動運転を実現するためには誤差の少ない自己位置推定や周辺環境の認知が重要となる．そのために考えられているのが，ダイナミックマップ [2], [3] である．ダイナミックマップとは，静的なデータである高精度地図をベースに，渋滞情報や事故による通行規制，センサから得られた交通データなどの動的な位置情報を組み合わせたものである．動的な位置情報が必要な理由として，単体の車両によるセンサのみでは認識できる範囲は非常に限定的で手前の物体に遮られると奥の物体が検出できなくなってしまうという点や，渋滞情報などから適切な走行ルートを選択することができるようになるという点などがあげられる．また，ダイナミックマップのベースとなる高精度地図 [4], [5] は従来のナビゲーション用の地図よりも細かい粒度となっている．従来のナビゲーション用の地図は各交差点をノード，交差点間を結ぶ道路を 1 本のリンクとして表現している．一方，高精度地図は各道路のレーン（車線）1 本 1 本について区別された詳細度をもっており，なおかつナビゲーション用の地図より精度の高いものとなっている．車線変更や道路の合流を支援するためには，各レーンについての情報が必要不可欠となる．

また，自己位置推定を行うための手法の 1 つである NDT スキャンマッチング [6], [7] では，点群データが必要とされている．点群データとは，3 次元の点の集合である．自動運転では周囲の環境を計測するために多くのセンサーを積んでおり，その内の 1 つに LiDAR（Light Detection and Ranging）がある．LiDAR はレーザーの反射を利用して点群データを集める．自己位置推定は，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照合することによって行われている．

名古屋大学発自動運転の実験システム Autoware [8] は Linux と ROS [9] をベースにしたシステムで, レーザーやカメラ，GNSS などのセンサー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．本研究で注目するのは，そのうちの 1 つにある自己位置推定である．Autoware では，ファイルに保存されている事前に作成した点群を，最初に一括で読み込んでメモリ上に展開し，LiDAR によって動的に測量した点群データと照合することで自己位置推定をおこなっている．

しかし，点群データは膨大である．データ点数および量は非常に大きく，例えば名古屋大学内のみで約 7,800 万点，10GB に達する．よって，大領域で自動運転を行う際に全ての点群をあらかじめ読み込んでメモリ上に展開する，という考えは現実的ではない．また，必要な点群のみを読み込むとはいってもファイルに保存されている状態では必要な点群を検索をすることは難しい．したがって，点群データは DBMS によって管理し，必要に応じてクエリによって取得する必要があると考えられる．また，自動運転はリアルタイムに行われているため，点群の検索の遅れによる自己位置推定の失敗は事故に繋がる危険がある．DBMS に格納した点群を検索するためには，その都度空間演算を行わなければならないが，空間演算には時間がかかる．そこで本研究では，高精度地図データと点群データを DBMS に取り込んで，高精度地図データを利用することで，点群データを効率良く検索する手法について検討することを目的とする．

本研究では，オープンソースソフトウェアであり RDBMS の 1 つである PostgreSQL [10] とその拡張機能で空間データを扱うことができる PostGIS [11] に高精度地図データおよび点群データを取り込む．そして，本研究で提案する「関連付け」を行い，あらかじめ点群の ID と高精度地図データの ID の対応

関係を作る．その対応関係を「関連テーブル」に記録する．作成した「関連テーブル」を利用することで，空間演算を行うことなく，レーンに沿った必要となる点群データのみを高速に取得することができるようになる．

本稿の以降の構成は以下のとおりである．第 2 節では，点群の管理に関連する研究について紹介する．第 3 節では，利用を想定したシステム Autoware とそこで用いられるデータについて述べる．第 4 節では，本研究で提案する手法について述べる．第 5 節では，提案手法の実験結果について述べる．最後に第 6 節で，本研究のまとめと今後の課題を述べる．

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．Oosterom ら [12] は，点群のロードや検索の比較を PostgreSQL，Oracle，MonetDB のデータベースごとに比較している．PostgreSQL における点群データの管理方法は 2 つ紹介されており，1 つは 1 行に 1 点を保存する普通のテーブルで，もう 1 つはいくつかの点をブロックにまとめて 1 行に 1 ブロックを保存するテーブルとなっている．後者の実現には Ramsei が開発した PostgreSQL の pointcloud 拡張機能 [13] のデータ型 PcPatch が使われている．この論文で，使用した点群は AHN2 [14] というものである．このデータはオランダ全土を対象に 1 平方メートルあたり 6〜10 個の点をもつデータセットとなっている．管理の対象となった点群データは自動運転をするためのデータではないという点や高精度地図と点群データの関係性は考えられていないという点が本研究と異なる．

3 Autoware における自己位置推定と用いられるデータ

本節では，自動運転システム用オープンソースソフトウェアである Autoware とそこで行われる自己位置推定のプロセスおよび用いられるデータである高精度地図データと点群データについて述べる．

3.1 Autoware

Autoware [8] は Linux と ROS [9] 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る．Autoware の全体像を図 1 に示す．レーザやカメラ，GPS などのセンサによって得られたデータをトピックに送信して，そのデータをトピックから受信することで自己位置推定（Localization）や障害物検知（Detection）を行い，それに基づいて経路計画（Mission）などをたてる．

Autoware では，運転前に csv 形式で保存されている高精度地図データと pcd 形式で保存されている点群データを読み込んでいる．点群データは図 2 のように分割されたメッシュごとにファイルで保存されている．そして，GPS や手動で入力した初期位置から大まかな位置情報（約 1〜10m の誤差）を得たうえで，事前に読み込んだ点群データと LiDAR によって動的に測量した点群データによる NDT スキャンマッチング [6], [7] により自己位置推定を行い，詳細な位置情報（約 10cm の誤差）を得ている．そして，高精度地図データをもとに経路探索などによって走行ルートを決定する．自己位置推定の流れを図 3 に示す．また，実験車両を図 4，自己位置推定の様子を図 5 に示す．図 5 の中の白い点は事前に読み込んだ点群データを表し，赤い点は LiDAR によって測量している点群データである．

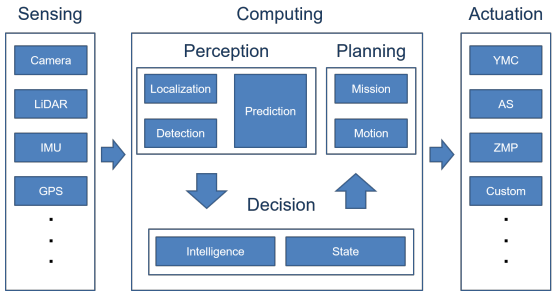

図 1　Autoware の概要（[15] より引用）

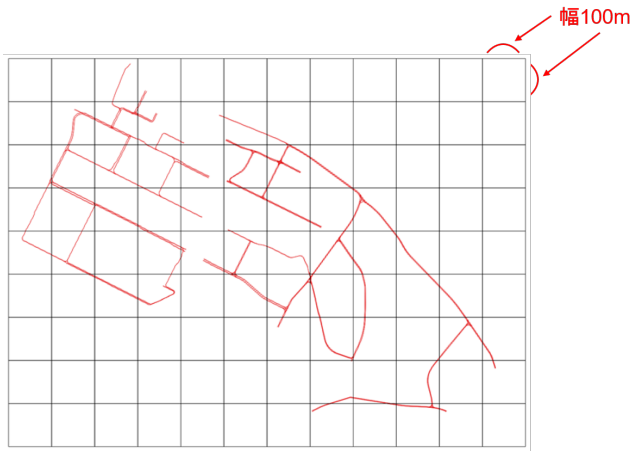

図 2　メッシュによる分割

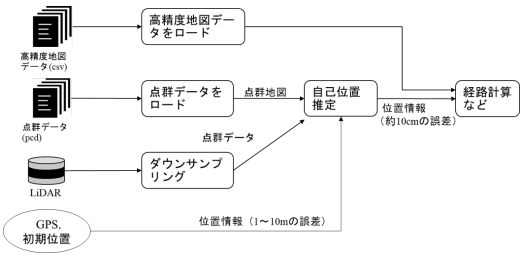

図 3　自己位置推定の流れ（[16] を参考に作成）

図 4　実 験 車 両

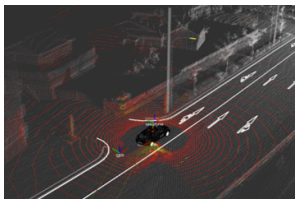

図 5　自己位置推定の様子

3.2 高精度地図データおよび点群データ

高精度地図データと点群データは Autoware で使用するデータである．高精度地図データおよび点群データは MMS（モービルマッピングシステム）によって収集，作成される．MMS は GPS や IMU, オドメータにより車両位置を高精度に取得し，同期した LiDAR およびカメラによって，走行しながら路面情報や道路周辺の環境を 3 次元データで高精度に取得できるシステムである [17]．本研究の実験では，PostgreSQL とその拡張機能であり空間データを扱える PostGIS にデータを取り込んでいる．

3.2.1 高精度地図データ

高精度地図はダイナミックマップのベースとなる静的な地図である．本研究で使用する高精度地図は、点，線，面によって構成されており，それらによって，「地物」や車の通り道を示す「レーン中心線」，「道路領域」などを表現している．本研究で点群データとの関連付けに利用する高精度地図データは，「レーン中心線」および「道路領域」である．データ型としては，PostGIS の拡張によって扱えるジオメトリ型における 3 次元の線，面である LINESTRINGZ，POLYGONZ を用いている．図 6 にテーブルの一部を示す.「レーン中心線」は線データ（linestring）であり，レーン中心線の ID（lnid），前のレーン中心線 ID（blid），次のレーン中心線 ID（flid）を保持している.「道路領域」は面データ（polygon）であり，道路領域の ID（waid）を保持している．名古屋大学内の「レーン中心線」，「道路領域」の一部を描画したものを図 7,8 に示す．

テーブル名: lane

lnid	linestring	blid	flid	…
832	LINESTRINGZ(…)	0	833	

テーブル名: wayarea

waid	polygon	…
66	POLYGONZ(…)	

図 6　レーン中心線と道路領域のテーブル

図 7　レーン中心線

図 8　道 路 領 域

3.2.2 点群データ

点群データは 3 次元情報を持つ点である．名古屋大学キャンパス前の点群を描画したものを図 9 に示す．点群データは csv 形式でファイルに保存されており，実験システム [8] では，これをバイナリ形式の pcd ファイル [18] に変換して最初に一括でメモリに読み込み，LiDAR によって動的に測定した点群と照合して自己位置推定を行っている．しかし，データ点数および量は名古屋大学内のみで約 7,800 万点，10GB に達するため，より大領域で実験を行うときには全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，必要な点群のみを状況に応じて読み込むことが必要となる．

これを PostgreSQL に取り込み，データ型として PostGIS の拡張によって扱えるジオメトリ型における 3 次元の点である POINTZ を用いる．図 10 にテーブルを示す．id は点群データに一意に割り振られた ID，point の（x,y,z）で x,y は平面直角座標系の x,y 座標，z は標高を表す．また，b,l,h は緯度，経度，楕円体高を表す．time は日曜日朝 9:00 からの通算秒数，error はレーザ照射時の車両位置に対する予測誤差，rd,gr,bl は RGB 値を表す．

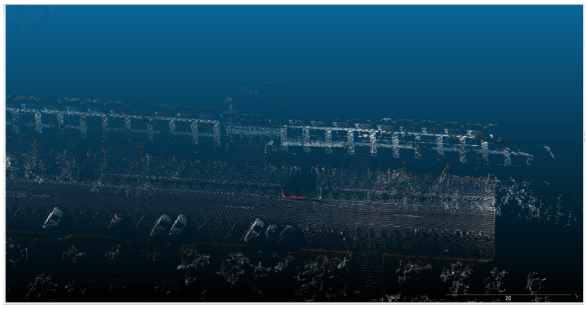

図 9　点群データの描画（名古屋大学キャンパス前）

テーブル名: point_cloud

id	point	b	l	h	time	error	rd	gr	bl
1	POINTZ(-18244.7 -93897.7 45.6)	35.153	136.97	83.971	525279.7	0.013	192	192	192

図 10　点群データのテーブル

4 提 案 手 法

4.1 概　　要

これまでに述べたように高精度地図データおよび点群データは DBMS によって管理する必要があると考えられる．そこで，図 11 に示すような点群・地図検索システムを提案する．入力として GPS などから得られた精度の低い位置情報を与え，それをもとに車両のいるレーン周辺の点群を検索する処理を行い，出力として検索によって得られた点群を返す．本研究では，Autoware から点群の検索が行われるという想定のもと，点群の効率のよい検索方法に焦点をあてる．

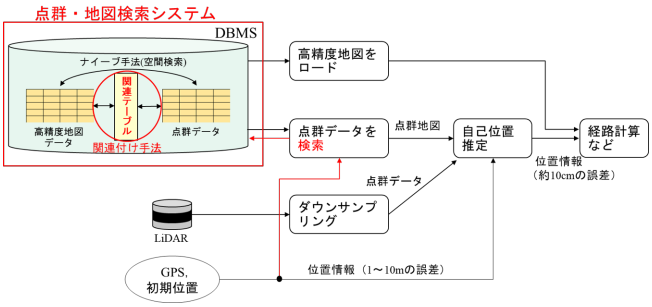

図 11　提案するシステムの全体像

単純な検索手法では，PostGIS によって拡張される機能を利用して，ある地点から一定距離内の点群を取得する距離検索やある領域と重なる点群を取得する範囲検索などの空間演算を行わなければならない．しかし，点群の検索は可能な限り高速な

ほうが望ましい．そこで本節では，点群の検索効率化の提案手法として，高精度地図データと点群データの「関連付け」について述べる.「関連付け」とは,「関連付け手法」を用いて，点群の ID と高精度地図データの ID の対応関係を作ることである．それを「関連テーブル」に記録しておき，点群データの検索にこの「関連テーブル」を利用することで，検索時には空間演算を行うことなく点群を検索することができる．

まず，空間データの検索に空間演算を行う単純な手法（以降，ナイーブ手法とする）の概要と問題点を述べる．

4.2 ナイーブ手法

ナイーブ手法では，空間データである点群の検索に空間演算を行う．空間演算は PostGIS による拡張機能として与えられる空間関係関数 [19] によって行われるものである．例えば ST_Dwithin(g1, g2, distance) 関数は，第 1 引数（g1）と第 2 引数（g2）のジオメトリ型の距離が第 3 引数（distance）で与えられる数値以内であるときに TRUE を返す関数である．この関数を使って，図 12 に示すように ID=1 の「道路領域」から 50m 以内にある点群を検索するといった距離検索が可能である．FROM 句の point_cloud は点群データのテーブル名，wayarea は「道路領域」のテーブル名を示す．WHERE 句で「道路領域」の ID である waid=1 を指定し，ST_Dwithin() 関数で距離 50m 以内の点群を検索することができる．しかし，空間演算には時間がかかってしまう問題がある．

```
SELECT point,rd,gr,bl
FROM point_cloud,wayarea
WHERE waid = 1 AND ST_Dwithin(polygon,point,50)
```

図 12　距離検索の SQL

4.3 関 連 付 け

本研究における「関連付け」とは，点群データの ID と高精度地図データの ID の対応関係を事前に作ることである．対応関係の作り方として「関連付け手法」を，対応関係の保存の方法として「関連テーブル」の構成方法をそれぞれ提案する．

4.3.1 関連付け手法

「関連付け手法」については「レーン全周囲型」と「レーン左右重視型」の 2 つを検討する．この 2 つはレーンと各点の対応付けの基準が異なっている．

「レーン全周囲型」

自己位置推定のときに必要な点群は，自車両のいる道路周辺の点群である．よって,「道路領域」から一定距離内の点群を効率良く取得できれば良いと考え，以下のように対応関係を作る．

- **点群と「道路領域」間の最短距離が R m 以内のもの同士が対応関係にあるとして「関連テーブル」に記録する．**

点群と「道路領域」の最短距離が R m 以内であるもの同士を「関連テーブル」に記録することによって「道路領域」周辺の点群を取得したいときには，その「道路領域」の ID（waid）をもとに検索を行うことができる．図 13 は点群データおよび道路領域データの一部を描画している．これは R=50 としたときの例であり，桃色の面は「道路領域」で，付近の 66，67 とあるのは waid である．また，青い点は点群データ，その付近の数字は点群に割りふられた id である．橙色の線は「道路領域」から距離 50m の線であり，この内側の点の id と道路領域の waid は対応関係にある．両矢印は，対応関係にある点群と「道路領域」を示しており，(id,waid) =（6802093,66）,（21968835,67）,（22292733,67）,（32439552,66）,（32439552,67）,（39292745,66）が「関連テーブル」に記録する組み合わせとなる．

本研究の実験では，名古屋大学内には建物が多いことから，LiDAR によって取得する点群と自車両の距離は 50m 以内になると考え，R=50 とする．

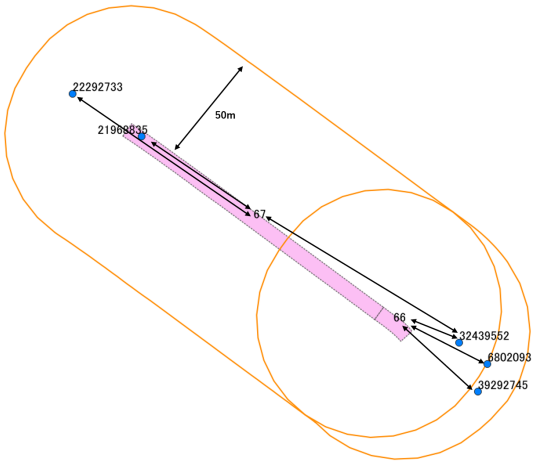

図 13　点群と「道路領域」の対応関係の例

「レーン左右重視型」

「レーン全周囲型」はレーンの少し先まで点群を取得できるが，その分だけ図 13 における id=32439552 の点のように，1 つの点に対応する「道路領域」が多く存在してしまい,「関連テーブル」のデータ量が多くなってしまうことが考えられる．したがって，この問題を解決するために以下のように対応関係を作る．

- **「レーン中心線」を基準に左右方向へ直角に R m 広げてできる領域を作り，その領域の元となる「レーン中心線」と領域に重なる点群を「関連テーブル」に記録する．**

「レーン中心線」を中心に作られる領域と点群を「関連テーブル」に記録することで，レーン中心線 ID（lnid）の指定によって「レーン中心線」周辺の点群を取得できるようになる．図 14 は点群データと「レーン中心線」,「レーン中心線」から作られる領域の一部を描画している．これは R=50 としたときの例であり，赤色の線は「レーン中心線」，緑色の面は「レーン中心線」から作られる領域である．また，付近の 832，999 は lnid を示しており，青い点は図 13 と同様に点群を示している．両矢印は対応関係にある点群と「レーン中心線」を示しており，(id,lnid) =（1457847,832）,（31128537,832）,（31445118,832）,（14127402,999）が「関連テーブル」に記録する組み合わせと

なる．

本研究の実験では,「レーン全周囲型」と同様の理由から R=50 とする．

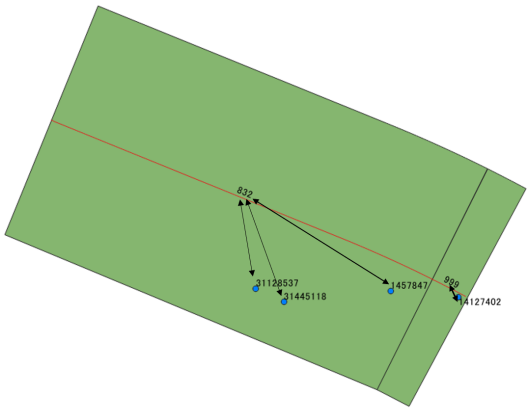

図 14　点群と「レーン中心線」から作られる領域の対応関係の例

それぞれの利点と欠点についてあげる.「レーン全周囲型」はレーンの少し先まで点群を取得できるが，1 つの点に対応する「道路領域」が多くなり,「関連テーブル」のデータが大きくなってしまうことがある．逆に「レーン左右重視型」は,「関連テーブル」のデータは小さくなるが，レーンの先の点群を取得するためには，次のレーン ID を通して取得する必要がある．

4.4 「関連テーブル」の構成方法

ID の対応関係を保存する形として「1:1 型」,「1:n 型」,「カラム追加型」の 3 つを検討する．3 つを検討する理由は，DBMS が配列をサポートしていない場合や，元データである点群データのテーブルに変更を加えられない場合も考慮する必要があるためである．

- 1:1 型

独立したテーブルに ID の対応関係を 1:1 で保存する．本研究の PostgreSQL を用いた実験では，btree インデックスを付与する．これは，配列を使用していないので，データ型として配列をサポートしていないデータベースでも利用できる．

- 1:n 型

独立したテーブルに ID の対応関係を 1:n（複数）で保存する．複数の ID の保持には配列を利用し，本研究の PostgreSQL を用いた実験では，gin インデックスを付与する．元データである点群データのテーブルに変更を加えられないときも利用できる．

- カラム追加型

点群データのテーブルのスキーマを拡張し，対応するレーンの ID を持つカラムを追加する．本研究の PostgreSQL を用いた実験では，追加したカラムには gin インデックスを付与する．カラム追加型は点群データのテーブルにそのまま関連テーブルを追加したので，テーブル間の結合を行うことなく道路領域 ID またはレーン中心線 ID の指定により，データを得ることができる．ただし，データ管理上の権限の制約などにより元のテーブルに変更を加えられない場合は，利用できない．

「関連付け手法」として「レーン全周囲型」を用いたときの「関連テーブル」3 種の構成の例を図 15 に示す．点群データと「道路領域」の対応関係を両矢印が示し，それによって作られる「1:1 型」,「1:n 型」,「カラム追加型」が表されている．

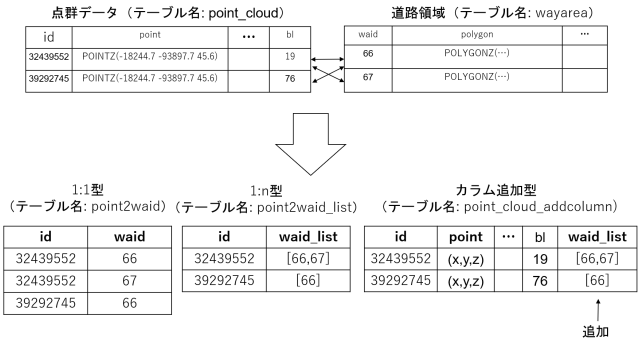

図 15　関連テーブル

4.5 関連テーブルを利用した点群データの検索

関連テーブルを用いた点群データの検索のやり方を示すため，ここではレーン全周囲型の場合でそれぞれの関連テーブルを扱う SQL を説明する．なお，それぞれの SQL において必要となる（x,y,z）座標と RGB 値を取得している．

- 「1:1 型」

関連テーブルとして「1:1 型」を利用したときの SQL を図 16 に示す．point2waid は「1:1 型」のテーブルを表し，WHERE 句の waid = {*waid*} で道路領域の ID（waid）の指定をし，点群データのテーブルと NATURAL JOIN を行うことで点群を検索する．

```
SELECT point, rd, gr, bl
FROM point_cloud NATURAL JOIN point2waid
WHERE waid = {waid}
```

図 16　1:1 型の SQL

- 「1:n 型」

関連テーブルとして「1:n 型」を利用したときの SQL を図 17 に示す．point2waid_list は「1:n 型」のテーブルを表し，WHERE 句の waid_list && array[{*waid*}] で道路領域の ID（waid）の指定をし，点群データのテーブルと NATURAL JOIN を行うことで点群を検索する．&&演算子は，配列に重複要素があるときに TRUE を返す．

```
SELECT point, rd, gr, bl
FROM point_cloud NATURAL JOIN point2waid_list
WHERE waid_list && array[{waid}]
```

図 17　1:n 型の SQL

- 「カラム追加型」

関連テーブルとして「カラム追加型」を利用したときの SQL を図 18 に示す．point_cloud_addcolumn は点群データのテーブルに対応する「道路領域」の ID を配列で保持したカラムを追加

したもので,「1:n 型」のときと同様に WHERE 句の waid_list && array[{*waid*}] で道路領域の ID（waid）の指定する．しかし,「1:1 型」と「1:n 型」と違い NATURAL JOIN を行う必要はない．

```
SELECT point, rd, gr, bl
FROM point_cloud_addcolumn
WHERE waid_list && array[{waid}]
```

図 18　カラム追加型の SQL

5　評 価 実 験

本研究で提案した「関連付け手法」と「関連テーブル」の構成方法の組み合わせについて，データサイズと検索効率の 2 つの観点から比較し，評価した．

5.1　実 験 環 境

実験環境を表 1 に示す．

表 1　PC の性能

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）

5.2　PostgreSQL の設定

文献 [20], [21] を参考とし，PC の性能をより発揮できるように postgresql.conf の設定を表 2 のとおり変更した．それ以外はデフォルトの設定とした．

表 2　変更した PostgreSQL の設定

shared_buffer	4GB	PostgreSQL 全体で使用する共有メモリバッファ
work_mem	1GB	各プロセスが使用するソート用のメモリバッファ
maintenance_work_mem	256GB	VACUUM，CREATE INDEX などで使用されるメモリの最大容量
effective_cache_size	16GB	OS と PostgreSQL のバッファキャッシュ内で利用可能と推定するメモリ量

5.3　データサイズの比較

点群データは膨大であるため，そこから作成した「関連テーブル」のデータサイズも大きくなる．データサイズが大きいとディスクを圧迫，またキャッシュによる高速化が行われにくくなるため，データサイズは小さいほうが望ましい．よって，作成した「関連テーブル」のインデックスサイズとテーブルサイズの合計を算出し，比較を行った．結果を図 19 に示す.「1:1 型」と「1:n 型」は関連テーブルのデータサイズを表し,「カラム追加型」は元データである点群のテーブルにカラムを追加したことによるデータサイズの増加量を表す．なお，元データである点群のデータサイズ（インデックスを含む）は約 16.393GB である．図 19 から分かることは以下のとおりである．

- 「関連テーブル」ごとのデータサイズを比較すると「レーン全周囲型」,「レーン左右重視型」どちらの「関連付け手法」もデータサイズの増加量は,「カラム追加型」＜「1:n 型」＜「1:1 型」となった．
  - 「カラム追加型」,「1:n 型」＜「1:1 型」

「カラム追加型」と「1:n 型」は，1 つの点に対応する道路領域またはレーン中心線がいくつあっても 1 つの点で 1 レコードとなることから，レコード数は点群の数と等しくなる．一方「1:1 型」は点群と道路領域またはレーン中心線との対応関係の数がそのままレコード数となるため，1 つの点に対応する道路領域またはレーン中心線が複数あるとき，その分だけ「カラム追加型」と「1:n 型」よりレコード数が増えてデータサイズが大きくなってしまう．そのため「カラム追加型」,「1:n 型」＜「1:1 型」となる．

  - 「カラム追加型」＜「1:n 型」,「1:1 型」

「カラム追加型」は既存のテーブルにカラムを追加するのに対して,「1:n 型」と「1:1 型」は新たにテーブルを作成するためだと考えられる．

- 同じ「関連テーブル」でも「レーン全周囲型」よりも「レーン左右重視型」のほうがデータサイズが小さい．
  - 1 つの点に対応する道路領域またはレーン中心線の数が「レーン全周囲型」よりも「レーン左右重視型」のほうが少ないためだと考えられる．レコード数が同じでも配列の要素数が多いほうがデータサイズが大きくなることが分かる．このことから,「関連付け手法」は，できる限り 1 つの点に対応する道路領域またはレーン中心線の重複の数が少なくなるようなものがよいと考えられる．

以上から，データサイズの観点では「関連付け手法」として「レーン左右重視型」を,「関連テーブル」として「カラム追加型」を利用するのが最もよいことが分かった．

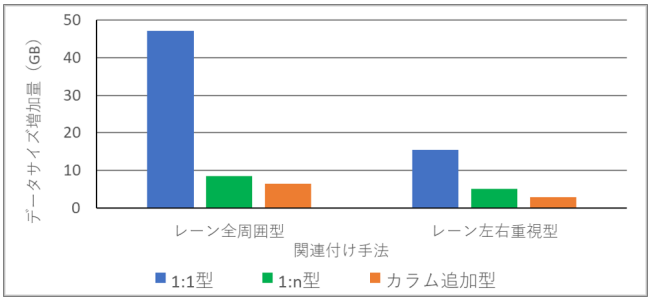

図 19　データサイズの増加量

5.4　ナイーブ手法および「関連テーブル」の違いによる検索時間の比較

「関連テーブル」を利用して点群を検索する SQL とナイー

ブ手法による SQL を実行し，その検索時間の測定を行った．検索によって取得した点群数とそれにかかった時間の関係を図 20 と図 21 に示す．なお，検索時間の測定には，EXPLAIN ANALYZE 句を用いた．また，全点群数は 78,249,885 点である．結果から分かることは以下のとおりである．

- 「レーン全周囲型」,「レーン左右重視型」どちらの「関連付け手法」も，クエリの検索時間は,「カラム追加型」＜「1:1 型」＜「1:n 型」＜「ナイーブ手法」となった．
  - 「1:1 型」＜「1:n 型」

配列を使った検索よりも使わない検索のほうが速いことがわかる．ただし，図 19 においてデータサイズは「1:1 型」のほうが大きいことから，取得する点群データの数がさらに多くなるとメモリに乗りきらなくなるため，キャッシュによる検索時間の高速化がきかなくなり「1:n 型」＜「1:1 型」となる可能性がある．

  - 「カラム追加型」＜「1:1 型」,「1:n 型」

テーブルの結合には時間がかかると考えられる．「カラム追加型」は，点群データのテーブルに対応する高精度地図の ID を保持するカラムを追加しているため，必要となるテーブルは点群データのもののみである．一方「1:1 型」と「1:n 型」は点群データと高精度地図データの ID の対応関係を別テーブルで保存しているので，点群を取得するためには，点群データのテーブルとの結合を行う必要がある．

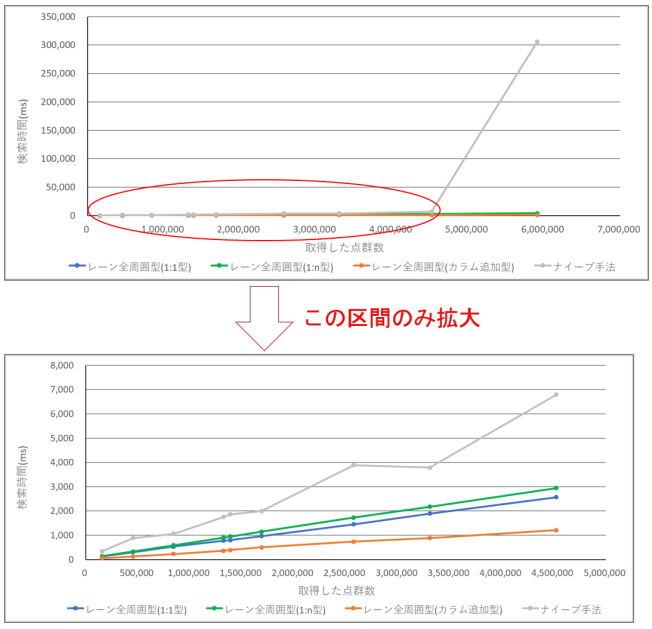

図 20 「レーン全周囲型」の検索時間

5.5 レーンごとの検索時間の比較

前小節で,「関連テーブル」を「カラム追加型」とするとき，検索時間が最も速くなることが分かった．Autoware では，点群はおよそ 100m 間隔のメッシュで分割されてファイルに保存されている．そこで，メッシュを用いた検索と本研究で提案した「関連付け」の比較を行う．

メッシュを用いた検索は、関連付け手法として 100m 間隔で区切ったメッシュの ID を点群データのテーブルにカラムとして追加する（関連テーブルは「カラム追加型」）．そしてメッシュの ID の指定によりメッシュ内の点群を取得できるようにする（以降，メッシュ法とする）．

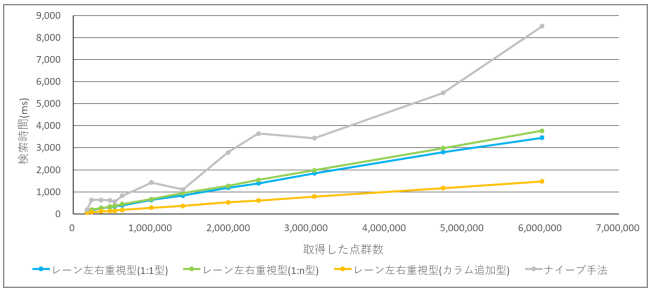

図 21 「レーン左右重視型」の検索時間

3 つのレーンを対象に「レーン全周囲型」,「レーン左右重視型」,「メッシュ法」の検索時間の測定をした．対象としたレーンを図 22 に示す．3 つのレーンを対象とした理由は，レーンとメッシュの関係による検索時間の変化を確認するためである．「レーン全周囲型」,「レーン左右重視型」は対象としたレーンのレーン中心線または道路領域の ID を指定することで,「メッシュ法」はレーンと重なるメッシュ内の ID を指定することで点群を取得する．結果を図 23 に示す．結果から分かることは以下のとおりである．

- 検索時間は「レーン左右重視型」＜「メッシュ法」＜「レーン全周囲型」である．
  - 「レーン全周囲型」と「レーン左右重視型」の検索時間の違いは，検索によって取得する点群数が「レーン全周囲型」のほうが多いために生じる．
  - 「レーン全周囲型」は取得する点群数が多いため,「メッシュ法」よりも検索時間が長くなった．
- レーンによって,「メッシュ法」と「レーン左右重視型」の検索時間の差が大きく異なる．
  - レーン A は，レーンの長さが小さく，必要となる点群が少ないが,「メッシュ法」では多くの点群を取得してしまう．
  - レーン B は，レーンが 2 つのメッシュに重なっているため,「メッシュ法」では余分に点群を取得してしまう．
  - レーン C は，レーンとメッシュが広く重なっているため,「メッシュ法」と「レーン左右重視型」の検索時間に大きな違いは見られない．

以上から,「レーン全周囲型」では「メッシュ法」より検索に時間がかかってしまうが,「レーン左右重視型」であれば「メッシュ法」よりも高速に点群を検索できることが分かった．また,「メッシュ法」による検索はレーンとメッシュの関係によっては必要のない点群も取得してしまう可能性が高いことが問題としてあげられる．

6 まとめと今後の課題

本研究では，自己位置推定に使われる点群データを無駄なく高速に取得するための手法を検討した．高精度地図データを利

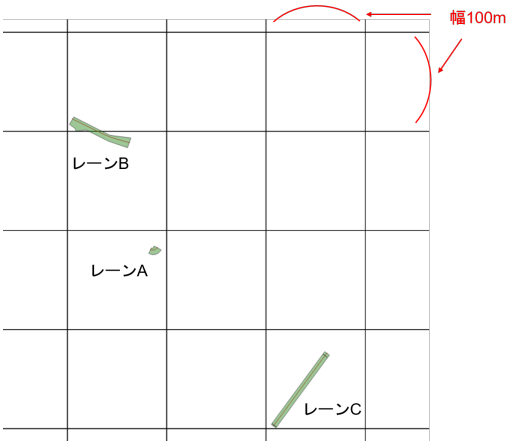

図 22　取り上げたレーンとメッシュの関係

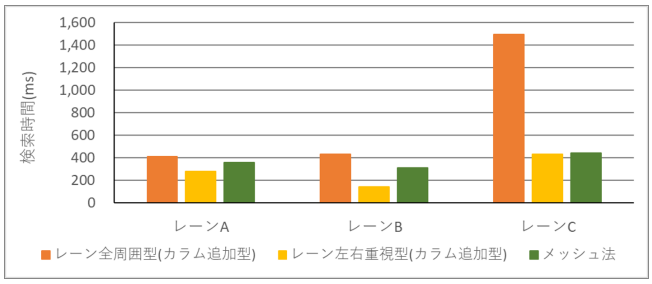

図 23　レーンごとの検索時間

用した「関連付け手法」を用いて「関連テーブル」を作成することによる点群データの検索効率化について検討し，実験を行った．結果，データサイズの増加量の観点からは「関連付け手法」は「レーン全左右重視型」,「関連テーブル」は「カラム追加型」がよいと分かった．検索時間の観点からも「関連付け手法」は「レーン全左右重視型」,「関連テーブル」は「カラム追加型」がよいと分かり，空間演算を行うよりも点群の検索が高速になることが確認できた．また，メッシュごとに点群を検索する「メッシュ法」との比較も行い，メッシュとレーンから必要となる点群がほぼ一致していない限りは,「レーン左右重視型」による検索のほうが高速となることが確認できた．

今後の課題としては，本研究で提案した「関連付け手法」において設定する距離が本実験で適用した 50m で適切であるのか，検索によって取得した点群を用いて自己位置推定を行うことができるのか調べていく必要があると考えられる．また，今回は高精度地図データを利用した点群データの検索効率化に焦点をおいたため，Autoware と DBMS の連携方法については考えていない．したがって，現段階では Autoware で DBMS を利用して点群を読み込ませることはできない．その連携方法を考えていき，さらには本研究で提案した手法を利用した実験を行うことも今後の課題としてあげられる．

謝　　辞

本研究の一部は，JST 産学共創プラットフォーム共同研究推進プログラム（OPERA）による．

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] 小山 浩，柴田 泰秀. “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.
[3] 渡辺 陽介，高木 健太郎，手嶋 茂晴，二宮 芳樹，佐藤 健哉，高田 広章. “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.
[4] “HERE HD Live Map — HERE - HERE Technologies”. https://www.here.com/products/automotive/hd-maps.
[5] “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.
[6] 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.
[7] 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.
[8] “自動運転ソフトウェア”. https://www.pdsl.jp/fot/autoware/.
[9] “Documentation - ROS Wiki”. http://wiki.ros.org.
[10] “PostgreSQL: The world’s most advanced open source database”. https://www.postgresql.org/.
[11] “PostGIS - Spatial and Geographic Objects for PostgreSQL”. https://postgis.net/.
[12] Peter van Oosterom, Oscar Martinez Rubi, Milena Ivanova, Mike Horhammer, Daniel Geringer, Siva Ravada, Theo Tijssen, Martin Kodde, Romulo Goncalves. “Massive point cloud data management: Design, implementation and execution of a point cloud benchmark”. *Computers & Graphics*, Vol. 49, pp. 92–125, 2015.
[13] “A PostgreSQL extension for storing point cloud (LIDAR) data”. https://github.com/pgpointcloud/pointcloud.
[14] “Actueel Hoogtebestand Nederland (AHN)”. http://www.ahn.nl/index.html.
[15] “Overview · CPFL/Autoware Wiki ·GitHub”. https://github.com/CPFL/Autoware/wiki/Overview.
[16] “自動運転/Autoware の特徴と最新動向”. https://www.scsk.jp/lib/product/oss/pdf/OSS_D3.pdf.
[17] “MMS（モービルマッピングシステム）”. http://www.whatmms.com/.
[18] “The PCD（Point Cloud Data）file format”. http://lang.sist.chukyo-u.ac.jp/Classes/PCL/PCDfileFormat.html.
[19] “空間関係関数と空間計測関数”. https://www.finds.jp/docs/pgisman/2.5.0/reference.html#Spatial_Relationships_Measurements.
[20] “Performance Optimization - PostgreSQL wiki”. https://wiki.postgresql.org/wiki/Performance_Optimization.
[21] 勝俣 智成，佐伯 昌樹，原田 登志. “[改訂新版] 内部構造から学ぶ PostgreSQL 設計・運用計画の鉄則”. 技術評論社, 2018.