ゼロから学ぶ LiDAR-Inertial Odometry と SLAM ～フルスクラッチで構築する 3 次元自己位置推定と地図生成～

赤井 直紀 [a, b]

[a] 株式会社 LOCT [b] 名古屋大学

ARTICLE HISTORY
Compiled June 9, 2025

1. はじめに

1.1. 背景知識

ロボットの自律移動や自動車の自動運転を行うにあたっては，地図を構築し，その地図上で走行している位置を認識する技術，いわゆる**自己位置推定**（Localization）や **Simultaneous Localization and Mapping**（SLAM）というよばれる技術が重要であるとされています．従来，これらの技術を用いる場合は，**オドメトリ**（Odometry）と呼ばれる移動量推定を行う枠組みが用いられていました．自己位置推定や SLAM で用いられている技術を端的に述べると，構築された地図（SLAM の場合はオンラインで構築している地図）と，センサの観測値を照合することで，地図上のどの位置に自分が存在しているかを認識する技術を用いています．この「センサと地図の照合」を行うにあたり，オドメトリから予測された移動量を用いることで，どの程度移動したかを予測することが可能となり，照合を行う際の探索範囲を限定することができるようになります．そのため，オドメトリを用いると精度や頑健性を向上させることができます．

しかしオドメトリを用いるとなると，自己位置推定や SLAM で用いられる外界センサ（LiDAR やカメラ）以外のセンサが必要になります．オドメトリシステムを構築する上で最も簡単な方法（システムを構築する手間が最もかからないという意味で）は，**Inertial Measurement Unit**（IMU）を用いることだといえます．IMU は，センサを原点とした加速度と角速度を計測できるセンサであり，これらの値を積分していくだけで移動量を計算することができます．しかし IMU の計測値が含む誤差は大きく，単に積分して得られた位置や角度の精度は極めて低く，自己位置推定や SLAM には利用できないことがほとんどです．そのため，IMU だけを用いてオドメトリシステムを構築することは不可能に近いといえます．

移動ロボットや自動運転の分野で最も広く使われているオドメトリシステムは，エンコーダ等を用いて車輪の回転量を計測し，その結果を積分することで移動量を計算する方法です．これは**ホイールオドメトリ**（Wheel Odometry）や**デッドレコニング**（Dead Reckoning）と呼ばれます．デッドレコニングは，タイヤの空転等が発生しない限り，短距離であれば移動量を正確に計測することができます．ただしデッドレコニングを用るには，外界センサだけでなく，移動体のハードウェアにも大きな変更を加える必要がります．そのためデッドレコニングは，安易に利用できるシステムとは言い難いです．またデッドレコニングは，車輪の回転量を計測することが前提のため，基本的に車輪型の移動体にしか適用することができません．

デッドレコニングに頼らない移動量の推定方法として，**ビジュアルオドメトリ**（Visual Odometry）が提案されました．ビジュアルオドメトリとは，画像から得られる特徴を追跡することで移動量を推定する方法です．そのためビジュアルオドメトリは，車輪型以外の移動体にも適用することが可能です．しかし一般に，ビジュアルオドメトリの精度はデッドレコニング程高くはないことが知られています．

ビジュアルオドメトリが提案された後に，LiDAR を用いて移動量推定を行う **LiDAR Odometry**（LO）も提案されました．LO では，LiDAR が計測する点群を逐次的に照合していくことで移動量の推定を行います．LiDAR の距離計測の精度は高いため，このような方法で移動量推定を行う LO の精度は，一般的にデッドレコニングの精度より高くなります．そのため LO は，様々な用途で使われるようになりました．

しかし LO にも弱点がありました．LiDAR，特に 3D LiDAR の計測周期は遅く（一般に 10 20 Hz 程度），高速な動き，特に回転を含む移動量を推定することは困難でした．この問題を解決する方法として提案されたのが，LiDAR と IMU を融合して移動量推定を行う **LiDAR-Inertial Odometry**（LIO）です．IMU は高周期（100 Hz 以上）で加速度と角速度を計測することができるため，LiDAR の計測周期の移動量を補間することができます．この移動の補間を用いると，高速に移動する LiDAR によって歪んでしまった LiDAR の計測点群を補正することができるようになります．また LIO では，

LO では推定されていなかった状態量（速度や IMU の計測値のバイアス）の推定も行います．そのため，IMU の計測値の積分も正確に行えるようになるため，移動量の推定をより高精度に行うことが可能になりました．

1.2. LIO の性能と限界

LIO のアルゴリズムの進化は目覚ましいものがありましたが，LiDAR の性能自体もここ数年で大きく進化しました．一昔前（著者が研究を始めたのが 2011 年）では,「LiDAR は価格コストが高いため，カメラを用いた手法を提案する」というのが論文等では常套文句でした．しかし今では，日本円で 10 万円程度で購入可能な 3D LiDAR も発売されています．そして驚くことに，このような価格の 3D LiDAR でも，360 度 100 m 程度のレンジを計測できるようになっており，LIO を用いて高精度な移動量推定を行うということはかなり一般的になってきました．そして LIO を用いるだけでも，小規模な環境であれば十分な精度の点群地図を構築することができるようになりました．そのため，ドローンのような飛翔体にこのような小型の LiDAR を搭載し，高精度な点群地図を生成することも容易に行われるようになってきました．

ただし LIO はあくまでオドメトリシステムであるため，移動量推定しか行いません．そのため，どれだけ高精度に移動量が推定できたとしても，推定量に誤差（ドリフト）が含まれてしまうため，LIO だけを用いて大規模な環境の地図構築を行うことはできません．特に大規模でループ（一度通過した地点を再度通過すること）が含まれる環境や条件ですと，整合性の取れた地図が構築できなくなってしまいます（同じものが同じ地点に正しくマッピングされなくなります）．前述した SLAM では，このようなループが含まれる場合であっても，整合性が取れた地図構築を行うことを目的としています．すなわち，精度の高い地図を構築したい場合には，SLAM の利用は避けられません．

また LIO はあくまで移動量推定のシステムです．応用上においては，移動量がわかるだけでは嬉しさがあることは少ないといえ，SLAM で構築した地図上で，どの位置にいるかを知れることの方が恩恵が多いといえます．例えば工場等で AGV やフォークリフトの位置を管理したい場合などには，LIO の利用だけでは不十分であり，自己位置推定の利用が求められます．そのため，単に高精度の LIO が利用可能になったというだけでは新たな応用システムを提案することは難しく，LIO に含まれるアルゴリズムを正しく理解し，それを SLAM や自己位置推定にも応用していくことが重要になります．

1.3. 既存手法と本書の立ち位置

LIO や SLAM を行うオープンソースはすでに多数存在しています．例えば LIO の有名なオープンソースとしては LIO-SAM や FAST-LIO が挙げられます．これらの手法の性能は極めて高く，これらをダウンロードして使用するだけでも，十分な移動量推定を行うことができるといえます．また LIO-SAM には SLAM の機能も含まれているため，地図構築を行うこともできます．また LiDAR SLAM の有名なオープンソースとしては，Cartographer や GLIM が挙げられます．これらの性能も極めて高く，様々な環境で極めて精度の高い点群地図を構築することができます．

しかし多くのソースコードは，機能を多く含むため，どうしても規模が大きくなってしまいます．そのため，初めて SLAM を学ぼうとする人がこれらを見ても，どこから何を追えば良いかの判断が難しく，結局ダウンロードして使うだけになってしまうことが多いと思います．本書，および対応するソースコードは，ソフトウェアの構成をとにかくシンプルに実装することに重きをおいています．開発したソースコードは，LIO, SLAM，自己位置推定の機能を有しており，その主な処理は空行を除いて 2000 行未満の C++で完結しています．この C++の中に，スキャンマッチング，LiDAR と IMU の融合（ルーズカップリングとタイトカップリング），ループ検知，ポーズグラフの最適

化といった必要な処理をすべて実装しています（用語の詳細は後ほど解説します）．また依存ライブラリも極力少なくし，ほぼフルスクラッチで LIO や SLAM を実装できるようになるようにしています．なお，主な依存ライブラリは Sophus（Eigen ベース）と nanoflann のみになります（他はパラメーター設定のために YAML を用いています）．これらはそれぞれ線形・Lie 代数を扱うライブラリと，最近某探索を行うライブラリとなっており，LIO や SLAM の根幹となる部分はすべてフルスクラッチで実装しています．

2. 数学的知識

2.1. 表記

本書では基本的に実数しか扱いません．そのため断りのない限り，すべて実数が使われることが前提になります．表記としては，スカラーを $a \in \mathbb{R}$，N 次元のベクトルを $\mathbf{a} \in \mathbb{R}^N$，$N \times M$ の行列を $A \in \mathbb{R}^{N \times M}$ と表記します．

2.2. ヤコビアン

本書では主に**最適化**（Optimization）を利用していきます．最適化とは，ある変数 $\mathbf{x} \in \mathbb{R}^N$ に従う関数 $f(\mathbf{x})$（最適化に使われる関数はおおよそ**コスト関数**（Cost Function）と呼ばれます）が定義されたときに，f の値を最小化（もしくは最大化）させること，またそれに対応する変数 $\mathbf{x}$ を求めることになります．最適化を行う方法には様々な方法がありますが，基本となることは，関数の勾配を求めることになります．

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_N} \end{pmatrix} \tag{1}$$

ベクトル関数 $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}) \cdots f_M(\mathbf{x}))^\top \in \mathbb{R}^M$ に関してもヤコビアンを定めることができます．

$$\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_N} \end{pmatrix} \tag{2}$$

なお本書では，基本的にヤコビアンを J と表記します．

ヤコビアンの計算は基本的に式 (1)，(2) に示す定義に従って行いますが，少し異なった方法でも導出することが可能です．まず，関数 $f(\mathbf{x})$ を $\delta\mathbf{x}$ だけ変化させた結果をテイラー展開を用いて近似します．

$$f(\mathbf{x} + \delta\mathbf{x}) \simeq f(\mathbf{x}) + J\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top H \delta\mathbf{x} \tag{3}$$

なお $H = \frac{\partial^2 f(\mathbf{x})}{\partial \mathbf{x}^2}$ であり，これは**ヘッセ行列**（Hessian）と呼ばれます．ここで，2 次の微小量を無視し，式 (3) の両辺が等しいと仮定すると，次式が得られます．

$$f(\mathbf{x} + \delta\mathbf{x}) - f(\mathbf{x}) = J\delta\mathbf{x} \tag{4}$$

すなわち，$f(\mathbf{x} + \delta\mathbf{x})$ と $f(\mathbf{x})$ の差分を $J\delta\mathbf{x}$ という形で記述できるとヤコビアンを求めることができます．

例えば，$f(x) = x^2$ という簡単な例を考えてみます．この関数のヤコビアンは $\frac{\partial x^2}{\partial x} = 2x$ となりますが，式 (4) に示す方法でヤコビアンを求めてみます．

$$\begin{aligned} &(x + \delta x)^2 - x^2 \\ =&x^2 + 2x\delta x + \delta x^2 - x^2 \\ =&2x\delta x \end{aligned} \tag{5}$$

ただし $\delta x^2 \simeq 0$ として 2 次の微小量を無視しました．式 (5) に示すように，$f(x) = x^2$ のヤコビアンを正しく求めることができました．この方法を用いると，関数を直接微分しなくてもヤコビアンを求めることができます．

2.3. ガウス・ニュートン法

本書で扱う問題は，度々最適化問題に帰着されます．最適化問題を解く際に用いられる方法は様々ありますが，本書では**ガウス・ニュートン法**（Gauss-Newton Method）を用います．

ガウス・ニュートン法を考えるにあたり，まず状態変数 $\mathbf{x} \in \mathbb{R}^N$，およびこの状態に依存する誤差ベクトル $\mathbf{e}(\mathbf{x}) \in \mathbb{R}^M$ を導入します．今，複数の誤差ベクトルを用いて，以下のコスト関数を定義します．

$$E = \sum_i \|\mathbf{e}_i\|_2^2 \in \mathbb{R} \tag{6}$$

ここで $\|\cdot\|_2^2$ は，ベクトルの 2 乗ノルムを計算する操作になります．そして，コスト関数を最小化する状態を以下のように定義します．

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\, E \tag{7}$$

これは，$\mathbf{x}$ の定義域でコスト関数を最小化する状態を $\mathbf{x}^*$ するという意味になり，このような解を**最適解**（Optimal Solution）と呼びます[1]．

ガウス・ニュートン法による最適化を考えるにあたり，まず誤差ベクトルを 1 次のテイラー展開で近似することを考えます．

$$\mathbf{e}(\mathbf{x} + \delta\mathbf{x}) \simeq \mathbf{e}(\mathbf{x}) + J\delta\mathbf{x} \tag{8}$$

ここで J は誤差ベクトル $\mathbf{e}$ の状態ベクトル $\mathbf{x}$ に関する偏微分 $\frac{\partial \mathbf{e}}{\partial \mathbf{x}} \in \mathbb{R}^{M\times N}$，すなわち**ヤコビアン**（Jacobian）になります．次に，式 (8) の近似された誤差ベクトルの 2 乗ノルムを考えます．

$$\begin{aligned}
&(\mathbf{e} + J\delta\mathbf{x})^\top(\mathbf{e} + J\delta\mathbf{x}) \\
=&(\mathbf{e}^\top + \delta\mathbf{x}^\top J^\top)(\mathbf{e} + J\delta\mathbf{x}) \\
=&\mathbf{e}^\top\mathbf{e} + \mathbf{e}^\top J\delta\mathbf{x} + \delta\mathbf{x}^\top J^\top\mathbf{e} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x} \\
=&\mathbf{e}^\top\mathbf{e} + 2J^\top\mathbf{e}\delta\mathbf{x} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}
\end{aligned} \tag{9}$$

なお，$\mathbf{e}^\top J\delta\mathbf{x} = \delta\mathbf{x}^\top J^\top\mathbf{e}$ を用いています．

続いて，式 (9) を $\delta\mathbf{x}$ の関数 $f(\delta\mathbf{x})$ とみなして，$\delta\mathbf{x}$ で偏微分します．

$$\frac{\partial f(\delta\mathbf{x})}{\partial \delta\mathbf{x}} = 2J^\top\mathbf{e} + 2J^\top J\delta\mathbf{x} \tag{10}$$

---

[1] しばしば,「良さそうな解」を「最適な解」と表現してしまう方が多いですが，工学的に最適な解とは最適解を意味し，最適解である以上あるコスト関数を最小化（もしくは目的関数を最大化）する解を意味してしまうことになります．最適という言葉を使うときは，必ず対応する関数を示すようにしてください．

そして，式 (10) が $\mathbf{0}$ になると仮定すると，次式が成り立ちます．

$$J^\top J\delta\mathbf{x} = -J^\top\mathbf{e} \tag{11}$$

ここで，式 (11) を満たす $\delta\mathbf{x}$ について考えます．$f(\delta\mathbf{x})$ は，$\delta\mathbf{x}$ だけ誤差ベクトルを変化させたものを近似し，その 2 乗ノルムを計算したものになっています．これを $\delta\mathbf{x}$ に関して偏微分し，その結果を $\mathbf{0}$ とすると，近似した誤差の 2 乗ノルムを最小にする $\delta\mathbf{x}$ を求めることが可能になります．すなわち，この操作で求められた $\delta\mathbf{x}$ 分だけ $\mathbf{x}$ を更新すると，コスト関数を減少させることができるようになります．

式 (9) を導出するにあっては，1 つの誤差ベクトルの近似を考えましたが，実際のコスト関数は複数の誤差ベクトルの 2 乗ノルムの和を計算しています．そのため，状態 $\mathbf{x}$ を $\delta\mathbf{x}$ だけ変化させたコスト関数を近似する必要がありますが，これは式 (12) のようになります．

$$E(\mathbf{x}+\delta\mathbf{x}) \simeq \sum_{i=1}^{N}\left(\mathbf{e}_i^\top\mathbf{e}_i + 2J_i^\top\mathbf{e}_i\delta\mathbf{x} + \delta\mathbf{x}^\top J_i^\top J_i\delta\mathbf{x}\right) \tag{12}$$

そして同様に，式 (12) を $\delta\mathbf{x}$ で偏微分した結果を $\mathbf{0}$ にすると，次式が得られます．

$$\sum_{i=1}^{N} J_i^\top J_i\delta\mathbf{x} = -\sum_{i=1}^{N} J_i^\top\mathbf{e}_i \tag{13}$$

ここで簡略化のため，以下のように変数を導入します．

$$\begin{aligned} H &= \sum_{i=1}^{N} J_i^\top J_i \\ \mathbf{b} &= \sum_{i=1}^{N} J_i^\top\mathbf{e}_i \end{aligned} \tag{14}$$

すなわち，$H\delta\mathbf{x} = -\mathbf{b}$ を満たす $\delta\mathbf{x}$ を得た後に，以下のように状態を更新します．

$$\mathbf{x} \leftarrow \mathbf{x} + \delta\mathbf{x} \tag{15}$$

なお，$H\delta\mathbf{x} = -\mathbf{b}$ からは，当然 $\delta\mathbf{x} = -H^{-1}\mathbf{b}$ が導けますが，$H \in \mathbb{N}\times\mathbb{N}$ になるため，N が大きい場合には逆行列の直接的な計算が困難になります．そのため，直接逆行列を計算することが少ないため，「$H\delta\mathbf{x} = -\mathbf{b}$ を満たす $\delta\mathbf{x}$」という表現を用いています．

2.4. リー群とリー代数

本書では頻繁に**リー群**（Lie Group）と**リー代数**（Lie Algebra）を用います．リー群やリー代数の厳密な説明は行いませんが，本書でリー群と呼ぶものは**回転行列**（Rotation Matrix）と**斉次行列**（Homogeneous Transformation Matrix）になります．回転行列は以下のように定義されます．

$$\{R \in \mathbb{R}^{3\times3} | R^\top R = I, \det(R) = 1\} \tag{16}$$

(17)

回転行列は Special Orthogonal Group in 3 Dimensions（SO(3)）とも呼ばれ，3 次元空間の回転を表現することができます．斉次行列は以下のように定義されます．

$$\left\{ \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} | R \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\} \tag{18}$$

斉次行列は Special Euclidean Group in 3 Dimensions（SE(3)）とも呼ばれ，3 次元空間での回転を含む位置（姿勢）を表現することができます．LIO や SLAM では基本的に，SO(3) や SE(3) を用いて状態を表現します．

リー群を用いる利点は，回転の状態を，途中で急に値が飛んだり切り替わったりすることなく，滑らかにかつ数学的に自然な形で扱えるということです．直感的には少し難しく感じるかもしれませんが，まずは平面上の回転，つまり xy 平面での角度 θ を例に考えてみます．角度 θ は通常，$0 \leq \theta < 2\pi$（あるいは $-\pi \leq \theta < \pi$）の範囲で定義されますが，$\theta = 0$ と $\theta = 2\pi$ は，数値としては異なるものの，回転としては同じ状態を表しています．このような性質から，角度 θ による表現では，状態が不連続に見えることがあります．しかしリー群を使えば，回転の変化を「切れ目のない空間」で表現でき，最初から最後まで滑らかに（連続的に）扱えるようになります．また，加減算や微分といった操作も数学的に統一されたルールで行えるので，処理が一貫してスムーズになります．

2.4.1. 反対称行列

ある 3 次元ベクトルに対して**反対称行列**（Skew-Symmetric Matrix）を生成する操作を以下のように定義します．

$$\mathbf{a}^\wedge = \begin{pmatrix} 0 & -a_z & a_2 \\ a_z & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \tag{19}$$

$$\begin{aligned} \theta &= \|\boldsymbol{\omega}\|_2 \\ \exp([\boldsymbol{\omega}]_\times) &= I_3 + \frac{\sin\theta}{\theta}[\boldsymbol{\omega}]_\times + \frac{1-\cos\theta}{\theta^2}[\boldsymbol{\omega}]_\times^2 \end{aligned} \tag{20}$$

$$\begin{aligned} \theta &= \arccos\left(\frac{\mathrm{tr}(R)-1}{2}\right) \\ \log(R) &= \frac{\theta}{2\sin\theta}\left(R - R^\top\right) \end{aligned} \tag{21}$$

なお $\log(R) = \boldsymbol{\omega}^\wedge \in \mathfrak{so}(3)$ となるため，この反対称行列から 3 次元ベクトル $\boldsymbol{\omega}$ を取り出す操作を以下のように定義します．

$$\boldsymbol{\omega} = \left(\boldsymbol{\omega}^\wedge\right)^\vee \tag{22}$$

3. スキャンマッチング

3.1. 概要

スキャンマッチングとは，2 つの点群を正しく照合できるような剛体変換 $T \in \mathrm{SE}(3)$ を求めることです．ここで T は，回転行列 $R \in \mathrm{SO}(3)$ と並進ベクトル $\mathbf{t} \in \mathbb{R}^3$ を含みます．

今，点群 $\mathcal{P} = (\mathbf{p}_1, ..., \mathbf{p}_N)$ と $\mathcal{Q} = (\mathbf{q}_1, ..., \mathbf{q}_M)$ を照合させることを考えます．ただし $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$ です．このとき，以下のコスト関数を考えます．

$$E = \sum_{i=1}^{N} \|\mathbf{q}_i - (R\mathbf{p}_i + \mathbf{t})\|_2^2 \tag{23}$$

ここで $\mathbf{q}_i$ は，$\mathbf{p}_i$ を剛体変換した点 $R\mathbf{p}_i + \mathbf{t}$ に最も近い $\mathcal{Q}$ 内の点です．式 (23) は，以下のように書くことも可能です．

$$E = \sum_{i=1}^{N} \|\mathbf{q}_i - T\mathbf{p}_i\|_2^2 \tag{24}$$

なおこの場合，$\mathbf{p}, \mathbf{q} \in \mathbb{R}^4$ となり，それぞれの 4 要素目には 1 が入ることになります．そのため，$T\mathbf{p}$ も 4 要素目が 1 の 4 次元ベクトルになりますが，$\mathbf{q}$ の 4 要素目も 1 となるため，$\mathbf{q} - T\mathbf{p}$ の 4 要素目は常に 0 になることととなり，結果としてコスト関数の値は式 (23) に示す値と同じになります．なお以下では，$\mathbf{q} - T\mathbf{p}$ を誤差ベクトル $\mathbf{e}$ として定めます．

スキャンマッチングでは，以下に示す剛体変換を求めることを考えます．

$$T^* = \underset{T}{\operatorname{argmin}}\, E \tag{25}$$

式 (25) は，コスト関数 E を最小化する姿勢 T^* を求めるという意味になります．この T^* は，一般的に反復処理を行うことで求めるられるため，Iterative Closest Points（ICP）スキャンマッチングとも呼ばれます．なお式 (24) に示すコストを最小にするスキャンマッチングは，対応する点同士の距離を最小にするため，point-to-point ICP とも呼ばれます．

point-to-point ICP は一般的にノイズに脆弱であるといわれています．そのため本書では，より頑健性の高い点と面の距離を最小化する point-to-plane ICP について考えます．point-to-plane ICP では，以下のコスト関数を考えます．

$$E = \sum_{i=1}^{N} \left(\mathbf{n}_i^\top \mathbf{e}_i\right)^2 \tag{26}$$

ここで $\mathbf{n}_i^\top$ は，$\mathbf{q}_i$ の周辺の点を用いて計算した面の 3 次元空間内での法線ベクトルです．なお，$\mathbf{e}$ が 4 次元ベクトルであるため $\mathbf{n}$ も 4 次元ベクトルとなりますが，$\mathbf{e}$ の 4 要素目は常に 0 であるため，$\mathbf{n}$ の 4 要素目がいくつであっても計算に違いは表れません．

3.2. ヤコビアンの計算

本書では，式 (26) に示すコスト関数の最小化を行うために，ガウス・ニュートン法を用います．そのために，残差 $r = \mathbf{n}^\top \mathbf{e}$ の姿勢 T に関するヤコビアンを求めます．この

ヤコビアンは，連鎖則を用いて以下のように計算できます.

$$\frac{\partial r}{\partial T} = \frac{\partial r}{\partial \mathbf{e}} \frac{\partial \mathbf{e}}{\partial T} \tag{27}$$

ここで，$\frac{\partial r}{\partial \mathbf{e}}$ は明らかに $\mathbf{n}^\top$ です．そのため，$\frac{\partial \mathbf{e}}{\partial T}$ についてのみ詳細の計算方法を示します.

残差ベクトル $\mathbf{e}$ は，明らかに姿勢 T に関する関数になっています．そのため，次の微小変化を考え，ることでヤコビアン J を導出します.

$$\mathbf{e}(T \oplus \delta T) - \mathbf{e}(T) = \mathbf{e}(\exp(\delta \boldsymbol{\xi}) T) - \mathbf{e}(T) \simeq J \delta \boldsymbol{\xi} \tag{28}$$

$\delta \boldsymbol{\xi}^\top = \begin{pmatrix} \delta \mathbf{t}^\top & \delta \boldsymbol{\theta}^\top \end{pmatrix}^\top \in \mathfrak{se}(3)$

$$\begin{aligned}
&\mathbf{q} - \exp(\delta \boldsymbol{\xi}) T \mathbf{p} - (\mathbf{q} - T\mathbf{p}) \\
=& - \left(\exp(\delta \boldsymbol{\xi}) - I_4\right) T \mathbf{p} \\
=& - \left(\begin{pmatrix} I_3 + [\delta \boldsymbol{\theta}]_\times & \delta \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} - I_4 \right) \begin{pmatrix} R\mathbf{p} + \mathbf{t} \\ 1 \end{pmatrix} \\
=& - \begin{pmatrix} [\delta \boldsymbol{\theta}]_\times & \delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}' \\ 1 \end{pmatrix} \\
=& - \begin{pmatrix} [\delta \boldsymbol{\theta}]_\times \mathbf{p}' & \delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \\
=& \begin{pmatrix} [\mathbf{p}']_\times \delta \boldsymbol{\theta} & -\delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \\
=& \begin{pmatrix} -I_3 & [\mathbf{p}']_\times \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \begin{pmatrix} \delta \mathbf{t} \\ \delta \boldsymbol{\theta} \end{pmatrix} = J \delta \boldsymbol{\xi}
\end{aligned} \tag{29}$$

なお $\mathbf{p}' = R\mathbf{p} + \mathbf{t}$ と置き，$[\delta \boldsymbol{\theta}]_\times \mathbf{p}' = -[\mathbf{p}']_\times \delta \boldsymbol{\theta}$ を用いました．よって，残差に対するヤコビアンは以下となります.

$$\begin{aligned}
\frac{\partial r}{\partial T} =& \mathbf{n}^\top \begin{pmatrix} -I_3 & [\mathbf{p}']_\times \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \\
=& \begin{pmatrix} -\mathbf{n}^\top & \mathbf{n}^\top [\mathbf{p}']_\times \end{pmatrix} \in \mathbb{R}^{1 \times 6}
\end{aligned} \tag{30}$$

ただし最後の $\mathbf{n}^\top$ は 3 次元の法線ベクトルになります.

$$\begin{aligned}
H &= \sum_{i=1}^{N} J_i^\top J_i \\
\mathbf{b} &= \sum_{i=1}^{N} J_i^\top r_i
\end{aligned} \tag{31}$$

[1].

ACKNOWLEDGMENT

This work was supported by KAKENHI under Grant 23K03773.

References

[1] J. Borenstein, H.R. Everett, L. Feng, and D. Wehe. Mobile robot positioning: Sensors and techniques. *Journal of Robotic Systems*, 14(4):229–340, 1997.