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 3, 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においては，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し，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．

しかしオドメトリを用いるとなると，自己位置推定や SLAM で用いられている外界センサ（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，単に積分するだけでは全く持って役に立たない精度でしか移動量を推定することができません．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です．

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予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あるため，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，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**）が提案されました．LO では，LiDAR が計測する点群を逐次的に照合していき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 LiDAR オドメトリ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，移動量の推定がより高精度に行うことが可能になります．

1.2. LIO の性能と限界

LIO のアルゴリズムの進化の他にも，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が，今では日本円で 10 万円を切る 3D LiDAR も発売されています．またこのような価格の 3D LiDAR でも，360 度 100 m 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レベルの点群地図を構築することができるようになりました．

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．前述した SLAM では，このようなループが含まれる条件であっても，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．

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. 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 $\mathbf{x} \in \mathbb{R}^N$ を導入します．そしてこの状態に依存する誤差ベクトル $\mathbf{e}(\mathbf{x}) \in \mathbb{R}^M$ も導入します．

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．

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

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 (1) は，以下のように書くことも可能です．

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

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 (1) 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．

$$T^* = \underset{T}{\operatorname{argmin}}\, E \tag{3}$$

式 (3) 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 (2) 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．

$$E = \sum_{i=1}^{N} \left(\mathbf{n}_i^\top \mathbf{e}_i\right)^2 \tag{4}$$

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 (4) 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．

$$\frac{\partial r}{\partial T} = \frac{\partial r}{\partial \mathbf{e}} \frac{\partial \mathbf{e}}{\partial T} \tag{5}$$

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．

$$\mathbf{e}(T \oplus \delta T) - \mathbf{e}(T) = \mathbf{e}(\exp(\delta \boldsymbol{\xi}) T) - \mathbf{e}(T) \simeq J \delta \boldsymbol{\xi} \tag{6}$$

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

$$\begin{aligned}
&\mathbf{q} - \exp(\delta \boldsymbol{\xi}) T \mathbf{p} - (\mathbf{q} - T\mathbf{p}) \\
=& - \left(\exp(\delta \boldsymbol{\xi}) - I_4 \right) T \mathbf{p} \\
=& - \left(\begin{pmatrix} I_3 + [\delta \boldsymbol{\theta}]_\times & \delta \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} - I_4 \right) \begin{pmatrix} R\mathbf{p} + \mathbf{t} \\ 1 \end{pmatrix} \\
=& - \begin{pmatrix} [\delta \boldsymbol{\theta}]_\times & \delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}' \\ 1 \end{pmatrix} \\
=& - \begin{pmatrix} [\delta \boldsymbol{\theta}]_\times \mathbf{p}' & \delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \\
=& \begin{pmatrix} [\mathbf{p}']_\times \delta \boldsymbol{\theta} & -\delta \mathbf{t} \\ \mathbf{0}^\top & 0 \end{pmatrix} \\
=& \begin{pmatrix} -I_3 & [\mathbf{p}']_\times \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \begin{pmatrix} \delta \mathbf{t} \\ \delta \boldsymbol{\theta} \end{pmatrix} = J \delta \boldsymbol{\xi}
\end{aligned} \tag{7}$$

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．

$$\begin{aligned}
\frac{\partial r}{\partial T} =& \mathbf{n}^\top \begin{pmatrix} -I_3 & [\mathbf{p}']_\times \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \\
=& \begin{pmatrix} -\mathbf{n}^\top & \mathbf{n}^\top [\mathbf{p}']_\times \end{pmatrix} \in \mathbb{R}^{1 \times 6}
\end{aligned} \tag{8}$$

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．

$$\begin{aligned}
H &= \sum_{i=1}^{N} J_i^\top J_i \\
\mathbf{b} &= \sum_{i=1}^{N} J_i^\top r_i
\end{aligned} \tag{9}$$

[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.