姿勢表現に回転ベクトルを用いた三次元グラフベースSLAM

○入江 清，友納 正裕（千葉工業大学）

3D Graph-based SLAM Using Rotation Vectors for Attitude Representation

○ Kiyoshi Irie and Masahiro Tomono (Chiba Institute of Technology)

Abstract : This paper presents our compact and portable implementation of 3D graph-based SLAM. We employ rotation vectors for state representation and optimization. One of the advantages of our method is high portability because it depends solely on Eigen algebra library. We compared the performance of our method with g^{2}o, a popular graph-based SLAM implementation, using three public datasets. Our method exhibited practical performance.

1. 緒言

Graph-based SLAM はロボットの地図構築における重要な技術の一つである．公開されている Graph-based SLAM の実装としては g^{2}o [1] が広く知られているが，依存するライブラリが多く，移植性が低いという問題がある．

これに対して，筆者らは文献 [2] において，線形代数ライブラリ Eigen を利用した二次元 Graph-based SLAM の実装を提案した．本論文ではそれを三次元に拡張する．本手法でも依存するライブラリは Eigen のみであり，従来手法の長所である移植性の高さを引き継ぐ．

本論文では，二次元 Graph-based SLAM を三次元へ拡張する上での主要な課題には誤差関数の定義とそのヤコビ行列の導出がある．三次元姿勢の表現には幾つかの選択肢があるが，本論文では姿勢の表現として回転ベクトル（rotation vector [3]）を用いた誤差関数の定義とそのヤコビ行列の詳細な導出を行う．また，Eigen を用いた具体的な実装方法についても紹介する．

実験として公開データセットを用いた速度評価を行い，提案手法が実用的な処理速度を有することを示す．

2. Graph-based SLAM の概要

Graph-based SLAM はロボットの位置姿勢列などをグラフとみなし，最適化によって修正する手法である．その基本的な考え方は Lu and Milios [4] によって最初に提案され，以降様々な変形版・改良版が提案されてきた [5] [6] [1]．ここでは Graph-based SLAM の定式化について述べる．

2.1 目的関数

ロボットやランドマークなどの位置姿勢を表すノードの列 $\{\boldsymbol{x}_i\}_{i=1}^n$ をノード間の拘束 $\{\boldsymbol{c}_{ij}\}_{<i,j>\in C}$ が与えられた下で最適化する．拘束は通常オドメトリや外界センサによる観測によって与える．

ノード列を表現する全パラメータを結合したベクトルを $\boldsymbol{x}$ と表記する．最適なノード列 $\boldsymbol{x}^*$ は下記の目的関数 F を最小化することにより求める．

$$\boldsymbol{x}^* = \min F(\boldsymbol{x}),$$
$$F(\boldsymbol{x}) := \sum_{<i,j>\in C} \boldsymbol{e}_{ij}(\boldsymbol{x})^\top \boldsymbol{\Sigma}_{ij}^{-1} \boldsymbol{e}_{ij}(\boldsymbol{x}).$$

ここで，$\boldsymbol{e}_{ij}$ は誤差ベクトル関数であり，具体的には $\boldsymbol{x}_i$ と $\boldsymbol{x}_j$ 間の相対位置姿勢と拘束によるノード間の相対位置姿勢 $\boldsymbol{c}_{ij}$ との差異を評価するものである．拘束に含まれる誤差は平均 0 の正規分布に従うものと仮定し，その共分散行列を $\{\boldsymbol{\Sigma}_{ij}\}_{<i,j>\in C}$ で表す．

2.2 最適化手法

目的関数は非線形であり，様々な非線形最適化手法が適用可能であるが，提案手法では Gauss-Newton 法を用いる．この手法はノード列の初期推定値 $\hat{\boldsymbol{x}}_{\text{init}}$ が最適解 $\boldsymbol{x}^*$ に十分近いことを前提とする．

Gauss-Newton 法は下記の 3 つの手続きを繰り返すことにより最適化を行う．

(1) 線形化: 目的関数の線形近似 $\bar{F}$ を計算する

(2) 最小化: $\bar{F}$ を最小化する差分ベクトル $\boldsymbol{d}$ を求める

(3) 更新: $\boldsymbol{d}$ によってノード列を更新する

以下に各ステップの詳細を説明する．

2.2.1 線形化

まず，目的関数 F を現在の推定値 $(\hat{\boldsymbol{x}})$ 周辺における線形近似 $\bar{F}$ を求める．全体の目的関数 F は各拘束によるコスト関数 $\boldsymbol{e}_{ij}$ の重み付き線形二乗和であり，$\boldsymbol{e}_{ij}(\boldsymbol{x})$ の $\boldsymbol{x}=\hat{\boldsymbol{x}}$ におけるヤコビ行列 $\boldsymbol{J}_{ij}$ を用いて下記のように近似する．

$$\begin{aligned} F(\hat{\boldsymbol{x}}+\boldsymbol{d}) &= \sum_{<i,j>\in C} \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}}+\boldsymbol{d})^\top \boldsymbol{\Sigma}_{ij}^{-1} \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}}+\boldsymbol{d}) \\ &\approx \sum_{<i,j>\in C} (\boldsymbol{e}_{ij}(\hat{\boldsymbol{x}})+\boldsymbol{J}_{ij}\boldsymbol{d})^\top \boldsymbol{\Sigma}_{ij}^{-1} (\boldsymbol{e}_{ij}(\hat{\boldsymbol{x}})+\boldsymbol{J}_{ij}\boldsymbol{d}) \\ &:= \bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d}). \end{aligned}$$

線形化されたコスト関数 $\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d})$ は下記の二次形式により表現される．

$$\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d}) = \boldsymbol{d}^\top \boldsymbol{H}\boldsymbol{d} + 2\boldsymbol{b}^\top \boldsymbol{d} + const., \tag{1}$$
$$\boldsymbol{H} := \sum_{<i,j>\in C} \boldsymbol{H}_{ij}, \quad \boldsymbol{b} := \sum_{<i,j>\in C} \boldsymbol{b}_{ij}$$
$$\boldsymbol{H}_{ij} := \boldsymbol{J}_{ij}^\top \boldsymbol{\Sigma}_{ij}^{-1} \boldsymbol{J}_{ij}, \quad \boldsymbol{b}_{ij} := \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}})^\top \boldsymbol{\Sigma}_{ij}^{-1} \boldsymbol{J}_{ij}.$$

2.2.2 最小化

式 (1) を最小化する $\hat{\boldsymbol{d}}$ は下記の線形一次方程式を解くことによって得られる．

$$\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{b}. \tag{2}$$

$\boldsymbol{H}$ はノード数が多いと巨大になるため，Gauss-Seidel 法のような単純な手法では大きな計算時間がかかるが，$\boldsymbol{H}$ は正定値対称行列であり，通常は疎であるため，コレスキー分解 [7] や共役勾配法 [8] が計算効率が良いことが知られている．

2.2.3 更新

最後に，得られた差分を用いて，下記のように現在の推定ノード列を更新する．

$$\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} + \hat{\boldsymbol{d}}. \tag{3}$$

ここまでの 3 段階の手続きを収束するまで繰り返す．

3. 回転ベクトル表現とその演算

本論文では誤差関数における三次元姿勢の表現方法として，回転ベクトル [3] を用いる．ここではその定義とヤコビ行列導出に必要な演算について説明する．

3.1 回転ベクトル表現

回転ベクトルは三次元のベクトル $\boldsymbol{v} = [v_1, v_2, v_3]^\top$ による三次元姿勢変換の表記方法である．$\boldsymbol{v}$ の値は長さ 1 の回転軸ベクトル $\boldsymbol{a} = [a_x, a_y, a_z]^\top$ および回転量 θ を用いて

$$\boldsymbol{v} := \theta \boldsymbol{a} \tag{4}$$

と定義する．逆回転は全要素の符号反転

$$\boldsymbol{v}^{-1} = -\boldsymbol{v} \tag{5}$$

によって得られる．

姿勢表現に回転ベクトルを用いる利点は，最小表現 (三次元) であること，オイラー角表現において問題となるジンバルロックが無いことがあげられる．欠点としては，回転の結合は簡単には計算できず，後に示すようにクォータニオンを経由したやや煩雑な計算が必要となる点がある．

以下では表記の簡素化のため $\boldsymbol{v}$ のノルムを

$$v := ||\boldsymbol{v}||$$

で表す．

3.2 単位クォータニオンとの変換

回転ベクトルの他に，広く用いられている三次元回転表現として単位クォータニオンがある．ある回転ベクトル $\boldsymbol{v}$ から同一の三次元回転を表すクォータニオンへの変換関数 $\boldsymbol{q}_v(\boldsymbol{v})$ は下記で与えられる．

$$\boldsymbol{q}_v(\boldsymbol{v}) := \begin{bmatrix} \cos\frac{v}{2} \\ \frac{v_1}{v}\sin\frac{v}{2} \\ \frac{v_2}{v}\sin\frac{v}{2} \\ \frac{v_3}{v}\sin\frac{v}{2} \end{bmatrix}. \tag{6}$$

ただし，

$$\boldsymbol{q}_v(\boldsymbol{0}) := [1, 0, 0, 0]^\top$$

である．

また，クォータニオンから回転ベクトルへの変換は

$$\boldsymbol{v}_q(\boldsymbol{q}) := \frac{2\arccos(q_0)}{\sqrt{1-q_0^2}} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{7}$$

によって与えられる．

$\boldsymbol{v}_q$ のクォータニオンに関する偏微分は下記のように計算される．

$$H(\boldsymbol{q}) := \frac{\partial \boldsymbol{v}_q(\boldsymbol{q})}{\partial \boldsymbol{q}} = \begin{bmatrix} 2cq_1(dq_0-1) & 2d & 0 & 0 \\ 2cq_2(dq_0-1) & 0 & 2d & 0 \\ 2cq_3(dq_0-1) & 0 & 0 & 2d \end{bmatrix}, \tag{8}$$
$$\text{ただし } c := \frac{1}{1-q_o^2}, \quad d := \frac{\arccos(q_0)}{\sqrt{1-q_o^2}}.$$

3.3 回転行列との変換

回転ベクトルから回転行列への変換を $\mathrm{R}(\boldsymbol{v})$ で表す．回転行列の回転ベクトルの各要素に関する偏微分は下記のようにクォータニオンを経由して計算できる．

$$\frac{\partial \mathrm{R}(\boldsymbol{v})}{\partial v_i} = \sum_{j=0}^{3} \frac{\partial \mathrm{R}_q(\boldsymbol{q}_v(\boldsymbol{v}))}{\partial q_j} G_{ji}(\boldsymbol{v}). \tag{9}$$

ここで，$G_{ji}(\boldsymbol{v})$ は $\boldsymbol{q}_v(\boldsymbol{v})$ の回転ベクトルに関するヤコビ行列 $G(\boldsymbol{v})$ の j 行 i 列目の要素を示す．$G(\boldsymbol{v})$ は下記のよ

うに計算される．

$$G(\boldsymbol{v}) := \frac{\partial \boldsymbol{q}_v(\boldsymbol{v})}{\partial \boldsymbol{v}} = \frac{S}{2v}\begin{bmatrix} -v_1 & -v_2 & -v_3 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} + \frac{a}{2v^3}\begin{bmatrix} 0 & 0 & 0 \\ v_1^2 & v_1v_2 & v_1v_3 \\ v_1v_2 & v_2^2 & v_2v_3 \\ v_1v_3 & v_2v_3 & v_3^2 \end{bmatrix}, \quad (10)$$

ただし　$S := \sin\frac{v}{2},\ a := v\cos\frac{v}{2} - 2S.$

また，R_q はクォータニオンから回転行列への変換であり，このクォータニオンによる偏微分は下記で与えられる．

$$\frac{\partial \mathrm{R}_q(\boldsymbol{q})}{\partial q_0} = 2\begin{bmatrix} q_0 & q_3 & -q_2 \\ -q_3 & q_0 & q_1 \\ q_2 & -q_1 & q_0 \end{bmatrix}, \quad (11)$$

$$\frac{\partial \mathrm{R}_q(\boldsymbol{q})}{\partial q_1} = 2\begin{bmatrix} q_1 & q_2 & q_3 \\ q_2 & -q_1 & q_0 \\ q_3 & -q_0 & -q_1 \end{bmatrix}, \quad (12)$$

$$\frac{\partial \mathrm{R}_q(\boldsymbol{q})}{\partial q_2} = 2\begin{bmatrix} -q_2 & q_1 & -q_0 \\ q_1 & q_2 & q_3 \\ q_0 & q_3 & -q_2 \end{bmatrix}, \quad (13)$$

$$\frac{\partial \mathrm{R}_q(\boldsymbol{q})}{\partial q_3} = 2\begin{bmatrix} -q_3 & q_0 & q_1 \\ -q_0 & -q_3 & q_2 \\ q_1 & q_2 & q_3 \end{bmatrix}. \quad (14)$$

3.4　回転ベクトルの積

2 つの回転ベクトル $\boldsymbol{v}, \boldsymbol{u}$ の積 $\boldsymbol{v} * \boldsymbol{u}$ は，回転の結合 ($\boldsymbol{v} \to \boldsymbol{u}$ の順) として定義する．この偏微分は連鎖律を用いて下記のように計算される [3].

$$\frac{\partial(\boldsymbol{v} * \boldsymbol{u})}{\partial \boldsymbol{v}} = H(\boldsymbol{q}_v(\boldsymbol{v})\boldsymbol{q}_v(\boldsymbol{u}))\bar{Q}(\boldsymbol{q}_v(\boldsymbol{u}))G(\boldsymbol{v}).$$

また，3 つの回転ベクトルの積に関する偏微分は

$$U_2(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) := \frac{\partial(\boldsymbol{u} * \boldsymbol{v} * \boldsymbol{w})}{\partial \boldsymbol{v}} = H(\boldsymbol{q}_v(\boldsymbol{u} * \boldsymbol{v} * \boldsymbol{w}))Q(\boldsymbol{q}^{\boldsymbol{u}})\bar{Q}(\boldsymbol{q}^{\boldsymbol{w}})G(\boldsymbol{v}), \quad (15)$$

$$U_3(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) := \frac{\partial(\boldsymbol{u} * \boldsymbol{v} * \boldsymbol{w})}{\partial \boldsymbol{w}} = H(\boldsymbol{q}_v(\boldsymbol{u} * \boldsymbol{v} * \boldsymbol{w}))Q(\boldsymbol{q}^{\boldsymbol{u}})Q(\boldsymbol{q}^{\boldsymbol{v}})G(\boldsymbol{w}), \quad (16)$$

と計算できる．ただし，$Q, \bar{Q}$ はクォータニオンの行列表現を表す．それぞれの具体的な計算手順は下記の通りである．

$$Q(\boldsymbol{q}) := \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix}, \quad (17)$$

$$\bar{Q}(\boldsymbol{q}) := \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix}. \quad (18)$$

$Q, \bar{Q}$ を用いてクォータニオン同士の積（クロス積）は下記のように計算できる[1]．

$$\boldsymbol{q}_a\boldsymbol{q}_b = Q(\boldsymbol{q}_a)\boldsymbol{q}_b = \bar{Q}(\boldsymbol{q}_b)\boldsymbol{q}_a. \quad (19)$$

4.　誤差関数とヤコビ行列

ここでは，Graph-based SLAM を三次元へ拡張する上での主要な課題である，誤差関数定義とそのヤコビ行列の導出を行う．

4.1　誤差関数の定義

提案手法では，各ノードの位置姿勢 $\boldsymbol{x}_i$ を並進に関して 3 次元，回転に関して 3 次元の合計 6 次元のベクトルで表現する．そのうち並進ベクトル成分を $\boldsymbol{t}^{(\boldsymbol{x}_i)}$，回転ベクトル成分を $\boldsymbol{v}^{(\boldsymbol{x}_i)}$ と表記する．

拘束 $\boldsymbol{c}_{ij}$ は $\boldsymbol{x}_i$ を基準とした $\boldsymbol{x}_j$ の相対位置姿勢として定義し，誤差関数を

$$\boldsymbol{e}_{ij}(\boldsymbol{x}+\boldsymbol{d}) := \begin{bmatrix} -\boldsymbol{t}^{(\boldsymbol{c}_{ij})} + \boldsymbol{t}^{(\Delta\boldsymbol{x}_{ij})} \\ (-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}) * \boldsymbol{v}^{(\Delta\boldsymbol{x}_{ij})} \end{bmatrix}, \quad (20)$$

ただし，$\Delta\boldsymbol{x}_{ij} := (\boldsymbol{x}_j \oplus \boldsymbol{d}_j) \ominus (\boldsymbol{x}_i \oplus \boldsymbol{d}_i)$

と定める．ここで $\oplus, \ominus$ は姿勢の合成演算子 [9] を表し，

$$\boldsymbol{a} \oplus \boldsymbol{b} := \begin{bmatrix} \boldsymbol{t}^{(\boldsymbol{a})} + \mathrm{R}(\boldsymbol{v}^{(\boldsymbol{a})})\boldsymbol{t}^{(\boldsymbol{b})} \\ \boldsymbol{v}^{(\boldsymbol{a})} * \boldsymbol{v}^{(\boldsymbol{b})} \end{bmatrix}, \quad (21)$$

$$\boldsymbol{a} \ominus \boldsymbol{b} := \begin{bmatrix} \mathrm{R}(-\boldsymbol{v}^{(\boldsymbol{b})})(\boldsymbol{t}^{(\boldsymbol{a})} - \boldsymbol{t}^{(\boldsymbol{b})}) \\ (-\boldsymbol{v}^{(\boldsymbol{b})}) * \boldsymbol{v}^{(\boldsymbol{a})} \end{bmatrix} \quad (22)$$

と計算される．

4.2　ヤコビ行列の導出

$\boldsymbol{e}_{ij}$ の $\boldsymbol{d} = \boldsymbol{0}$ におけるヤコビ行列は下記のような疎な構成となる．

$$\boldsymbol{J}_{ij} = \left(\cdots \boldsymbol{0} \cdots \boldsymbol{J}_{ij}^{(i)} \cdots \boldsymbol{0} \cdots \boldsymbol{J}_{ij}^{(j)} \cdots \boldsymbol{0} \cdots\right).$$

[1] 本論文ではクォータニオンの積の順序を文献 [3] とは逆順で定義しており，従って Q と $\bar{Q}$ の定義が入れ替わっている点に注意されたい．

ここで $\boldsymbol{J}_{ij}^{(i)}$ と $\boldsymbol{J}_{ij}^{(j)}$ は誤差関数 $\boldsymbol{e}_{ij}$ をそれぞれ $\boldsymbol{x}_i$ と $\boldsymbol{x}_j$ に関して偏微分することにより得られる行列である．それぞれの計算方法を下記に示す．

$$\begin{aligned}\boldsymbol{J}_{ij}^{(j)} &:= \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}+\boldsymbol{d})}{\partial \boldsymbol{x}_j}\Big|_{\boldsymbol{d}=0} \\ &= \frac{\partial}{\partial \boldsymbol{x}_j}\begin{bmatrix} -\boldsymbol{t}^{(\boldsymbol{c}_{ij})} + \boldsymbol{t}^{(\boldsymbol{x}_j \ominus \boldsymbol{x}_i)} \\ (-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}) * (-\boldsymbol{v}^{(\boldsymbol{x}_i)}) * \boldsymbol{v}^{(\boldsymbol{x}_j)} \end{bmatrix} \\ &= \frac{\partial}{\partial \boldsymbol{x}_j}\begin{bmatrix} \mathrm{R}(-\boldsymbol{v}^{(\boldsymbol{x}_i)})\boldsymbol{t}^{(\boldsymbol{x}_j)} \\ (-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}) * (-\boldsymbol{v}^{(\boldsymbol{x}_i)}) * \boldsymbol{v}^{(\boldsymbol{x}_j)} \end{bmatrix} \\ &= \begin{bmatrix} \mathrm{R}(-\boldsymbol{v}^{(\boldsymbol{x}_i)}) & \boldsymbol{0} \\ \boldsymbol{0} & U_3(-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}, -\boldsymbol{v}^{(\boldsymbol{x}_i)}, \boldsymbol{v}^{(\boldsymbol{x}_j)}) \end{bmatrix}, \end{aligned} \tag{23}$$

$$\begin{aligned}\boldsymbol{J}_{ij}^{(i)} &:= \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x}+\boldsymbol{d})}{\partial \boldsymbol{x}_i}\Big|_{\boldsymbol{d}=0} \\ &= \frac{\partial}{\partial \boldsymbol{x}_i}\begin{bmatrix} -\boldsymbol{t}^{(\boldsymbol{c}_{ij})} + \boldsymbol{t}^{(\boldsymbol{x}_j \ominus \boldsymbol{x}_i)} \\ (-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}) * (-\boldsymbol{v}^{(\boldsymbol{x}_i)}) * \boldsymbol{v}^{(\boldsymbol{x}_j)} \end{bmatrix} \\ &= \frac{\partial}{\partial \boldsymbol{x}_i}\begin{bmatrix} -\mathrm{R}(-\boldsymbol{v}^{(\boldsymbol{x}_i)})\boldsymbol{t}^{(\boldsymbol{x}_i)} \\ (-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}) * (-\boldsymbol{v}^{(\boldsymbol{x}_i)}) * \boldsymbol{v}^{(\boldsymbol{x}_j)} \end{bmatrix} \\ &= \begin{bmatrix} -\mathrm{R}(-\boldsymbol{v}^{(\boldsymbol{x}_i)}) & R'_1\Delta\boldsymbol{t}_{ij} & R'_2\Delta\boldsymbol{t}_{ij} & R'_3\Delta\boldsymbol{t}_{ij} \\ \boldsymbol{0} & -U_2(-\boldsymbol{v}^{(\boldsymbol{c}_{ij})}, -\boldsymbol{v}^{(\boldsymbol{x}_i)}, \boldsymbol{v}^{(\boldsymbol{x}_j)}) \end{bmatrix}, \end{aligned} \tag{24}$$

ただし $\Delta\boldsymbol{t}_{ij} := \boldsymbol{t}^{(\boldsymbol{x}_j)} - \boldsymbol{t}^{(\boldsymbol{x}_i)}$，$R'_k := -\dfrac{\partial \mathrm{R}(-\boldsymbol{v}^{\boldsymbol{x}_i})}{\partial v_k^{(\boldsymbol{x}_i)}}$.

5. 実装

ここでは，Eigen を用いた実装上の留意点について述べる．この内容は二次元と三次元とで共通である．

5.1 目的関数の構築

$\boldsymbol{J}_{ij}$ の疎な構造を利用すると，式 (1) における $\boldsymbol{H}_{ij}$ と $\boldsymbol{b}_{ij}$ はヤコビ行列 $\boldsymbol{J}_{ij}$ を明示的に構築することなく，下記のように計算できる．

$$\boldsymbol{H}_{ij} = \begin{pmatrix} \ddots & & & \\ & {\boldsymbol{J}_{ij}^{(i)}}^\top \Sigma_{ij}^{-1} \boldsymbol{J}_{ij}^{(i)} & \cdots & {\boldsymbol{J}_{ij}^{(i)}}^\top \Sigma_{ij}^{-1} \boldsymbol{J}_{ij}^{(j)} \\ & \vdots & & \vdots \\ & {\boldsymbol{J}_{ij}^{(j)}}^\top \Sigma_{ij}^{-1} \boldsymbol{J}_{ij}^{(i)} & \cdots & {\boldsymbol{J}_{ij}^{(j)}}^\top \Sigma_{ij}^{-1} \boldsymbol{J}_{ij}^{(j)} \\ & & & \ddots \end{pmatrix},$$

$$\boldsymbol{b}_{ij} = \begin{pmatrix} \vdots \\ {\boldsymbol{J}_{ij}^{(i)}}^\top \Sigma_{ij}^{-1} \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}}) \\ \vdots \\ {\boldsymbol{J}_{ij}^{(j)}}^\top \Sigma_{ij}^{-1} \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}}) \\ \vdots \end{pmatrix}.$$

上式において省略されている要素は全て 0 である．共分散行列は誤差関数と同じ座標系で表現する必要がある点に注意されたい．

$\boldsymbol{H}$ は $\boldsymbol{H}_{ij}$ の和により計算されるため，Eigen を用いた実装では，$\boldsymbol{H}$ を `Eigen::SparseMatrix` 型として定義し，$\boldsymbol{H}_{ij}$ の各非ゼロ要素を `Eigen::Triplet` に格納し足し合わせることによって構築することができる．

5.2 最小二乗解の安定化

式 (2) を解く際に，H が特異行列に近いと数値計算的に不安定になる場合がある．これを回避するため，H の対角要素に小さな非負数 λ を加え，

$$\boldsymbol{H} \leftarrow (\boldsymbol{H} + \lambda I) \tag{25}$$

とする．これはよく知られた安定化方法であり，リッジ回帰の正則化などでも同様の手法が用いられる [10]．なお，$\lambda > 0$ である場合の最適化は Levenberg-Marquardt 法と呼ばれる．

5.3 連立一次方程式ソルバ

Eigen には複数の連立一次方程式ソルバが含まれる．筆者らの実験ではコレスキー分解の変形の一つである $LDL^\top$ 分解 `Eigen::SimplicialLDLT` が高速であったため，これを採用する．

スパースコレスキー分解としては cholmod [11] が高速であることが知られており，Eigen には `CholmodSupport` という cholmod と接続するためのモジュールが提供されているが，筆者らの実験では速度の向上は見られなかった．

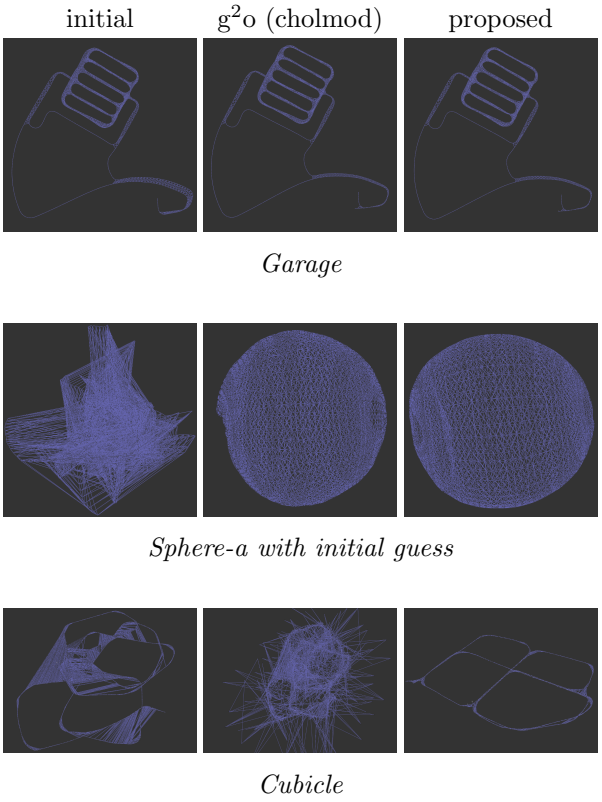
6. 処理速度評価

提案手法の処理速度を g^{2}o と比較した．g^{2}o の設定として，最適化手法には Gauss-Newton 法を，一次方程式のソルバには cholmod と CSparse [11] を使用した．入力データとしては，三種類の公開データセット Garage，Sphere-a および Cubicle を用いた [12]．ただし，Sphere-a は g^{2}o に実装されている Spanning tree に基づく初期値改善を適用してから使用した．これらのデータセットの概要を Table. 1 に示す．

それぞれのデータセットにおいて，各手法により最適化を行った結果を Fig. 1 に示す．提案手法はいずれのデータにおいても正しく形状を復元できていることが見て取れる．g^{2}o は Cubicle データではコレスキー分解処理に失敗し，復元を行うことができなかった．そのため，続く速度評価では Cubicle を除外する．

Table. 1: Overview of datasets used for evaluation.

Dataset	No. of poses	No. of constraints
Garage	1,661	6,275
Sphere-a	2,200	8,647
Cubicle	5,750	16,869



Garage

Sphere-a with initial guess

Cubicle

Fig. 1: Graph optimization results.

速度評価のため，1 回分の繰り返しに要した平均時間とそのうち一次方程式のソルバが消費した時間を測定した．Fig. 2 に比較のグラフを示す．

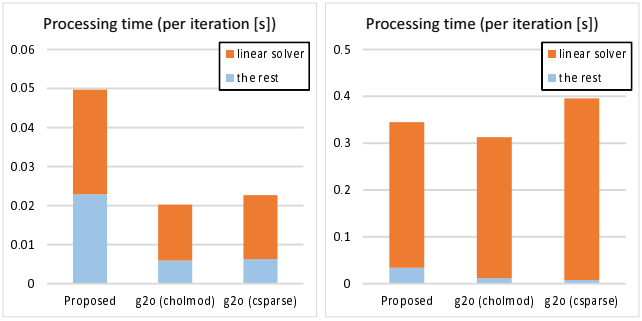
提案手法と g^{2}o の速度比はデータにより異なることが観察された．Garage では提案手法が 49.7ms であったのに対して g^{2}o (cholmod) は 22.4ms，g^{2}o (csparse) は 22.6ms であり，提案手法は g^{2}o の約 2.2 倍の処理時間であったが，Sphere-a では提案手法の 352ms に対して g^{2}o (cholmod) が 313ms，g^{2}o (csparse) は 395ms と差は小さい．

提案手法では，処理時間に占めるソルバによる求解時間以外（ヤコビアン計算，疎行列の構築など）の割合が g^{2}o に比べて多く，この点は実装上での最適化の余地があるものと考えられる．

7. 結言

本論文では Eigen を用いた三次元 Graph-based SLAM の実装を紹介した．三次元姿勢表現として回転ベクトルを用い誤差関数の定義とヤコビ行列の導出しを行った．

公開データセットにより処理速度の評価を行った結果，提案手法は g^{2}o には及ばないものの，実用的な速度を有するものと考えられる．今後の課題としては，提案手法の収束の速さや精度の評価などを行うことがあげられる．



Garage *Sphere-a*

Fig. 2: Comparisons of average processing time for one iteration. The processing times were measured on a Core i7-6950X.

参考文献

[1] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g^{2}o: A general framework for graph optimization. In *Proc. of ICRA*, pp. 3607–3613, 2011.

[2] K. Irie and M. Tomono. A compact and portable implementation of graph-based SLAM. In *Proc. of Robomech 2P2-B01*, 2017.

[3] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. Report, Stanford University, 2006.

[4] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, Vol. 4, pp. 333–349, 1997.

[5] E. Olson, J. Leonard, and S. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proc. of ICRA*, pp. 2262–2269, 2006.

[6] 竹内栄二朗, 坪内孝司. 疎行列連立方程式解法に基づく複数外界センサによる地図生成. 第 13 回ロボティクスシンポジア, pp. 166–173, 2008.

[7] K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Efficient sparse pose adjustment for 2d mapping. In *Proc. of IROS*, pp. 22–29, 2010.

[8] M. Montemerlo and S. Thrun. *Large-Scale Robotic 3-D Mapping of Urban Structures*, pp. 141–150. Springer Berlin Heidelberg, 2006.

[9] R. Smith, M. Self, and P. Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics*, pp. 167–193. Springer New York, 1990.

[10] A. E. Hoerl and R. W. Kennard. Ridge regression: Applications to nonorthogonal problems. *Technometrics*, Vol. 12, No. 1, pp. 69–82, 1970.

[11] Suitesparse : A suite of sparse matrix software. `http://faculty.cse.tamu.edu/davis/suitesparse.html`. [Accessed: 27- Sep- 2017].

[12] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert. Initialization techniques for 3D slam: A survey on rotation estimation and its use in pose graph optimization. In *Proc. of ICRA*, pp. 4597–4604, 2015.