Irie *Robomech J (2018) 5:14*
https://doi.org/10.1186/s40648-018-0110-1

ROBOMECH Journal

RESEARCH ARTICLE

Open Access

A graph optimization approach for motion estimation using inertial measurement unit data

Kiyoshi Irie*

Abstract

This study presents a novel approach for processing motion data from a six-degree-of-freedom inertial measurement unit (IMU). Trajectory estimation through double integration of acceleration measurements results in the generation and accumulation of multiple errors. Existing IMU-based measurement methods often use constrained initial and final states to resolve these errors. The constraints on the initial and final states lead to a uniform distribution of the accumulated errors throughout the calculated trajectory so that they cancel each other. We develop a generalized method that can incorporate the constraints from the measurements of intermediate states. The proposed approach is inspired by graph-based simultaneous localization and mapping processes from robotics research. We tested the proposed method with simulated and actual IMU data and found that our method estimates trajectories more accurately than conventional methods with acceptably higher computational costs.

Keywords: Motion estimation, Inertial measurement unit, Graph-based simultaneous localization and mapping

Introduction

Background

Measuring athlete motion is an important aspect of sports science. The typical instruments employed for this purpose include high-speed video cameras and infrared motion capture cameras. However, these instruments are expensive and require time and effort for installation.

Recently, small and inexpensive inertial measurement units (IMUs) have become available, making sports motion measurements accessible to a wide range of people. For example, measurement devices that can be attached to a golf club [1, 2] or a tennis racket [3] have been developed.

Considering these applications, this study seeks to improve the data-processing algorithm for extracting trajectories from IMU data. The IMU employed herein is assumed to comprise a three-axis accelerometer and a three-axis gyroscope. We aim to estimate a time series of the attitude, velocity, and position values with a time series of acceleration and angular velocity measurements.

Main issues to be addressed

The integration of IMU measurements is the simplest way to extract a trajectory from IMU data. Theoretically, attitude can be calculated by integrating the angular velocity, velocity can be calculated by integrating the acceleration, and position can be calculated by the double integration of the acceleration. However, position estimation by simple double integration amplifies the error [4]. Even a small error in the integration of the acceleration can lead to a bias drift in the velocity values. Additionally, the integration of biased velocity causes rapid generation of the positional errors. In addition, if the measurement is performed for a long period, the attitude and velocity estimations are affected by the bias drift in the angular velocity and acceleration values. Thus, it is necessary to correct the accumulated errors.

*Correspondence: irie@furo.org
Future Robotics Technology Center, Chiba Institute of Technology, 2-17-1, Tsudanuma, Narashino, Chiba, Japan

Related work

Human gait analysis is a well-studied topic in IMU-based motion estimation. Several methods that can precisely track human gait have been developed [5, 6]. Error correction methods for gait data rely on the assumption that the foot velocity is zero during the stance phase, which is the period during which the foot is in contact with the ground. The errors accumulated in the velocity data can be eliminated using this assumption [6, 7].

A method to correct motion errors without such zero-velocity assumptions needs to be developed because the assumptions are not always applicable to various sports motions. Sagawa et al. developed a method for measuring the pitching motion used in baseball [8]. They constrained the target motion with a known initial state and a known final state. The accumulated errors are corrected with regard to these constraints.

However, this method is limited as error correction is performed only with one constraint, thereby returning inaccurate trajectories for motions that extend over a long period of time. Therefore, a more powerful error correction method is needed to measure longer sets of motions, such as a rally in a tennis match.

Contribution

Herein, we propose a novel error correction method that can incorporate multiple constraints. While the relation between the initial and final states is the only constraint employed in the conventional motion estimation method, the proposed method employs generalized constraints that can represent the differences between any two known points in time. The problem is formulated as a least-squares fit of the constraint errors. Steps reducing the computational complexity of the algorithm are also introduced to make computations practical. We quantify the performance of the proposed algorithm with simulated and actual data.

IMU-based motion estimation: review

Problem definition

We consider the problem of estimating a time series of the states of an IMU.

$$\{\boldsymbol{x}_i\}_{i=1}^n, \quad \boldsymbol{x}_i := \begin{bmatrix} \boldsymbol{u}_i \\ \boldsymbol{v}_i \\ \boldsymbol{t}_i \end{bmatrix}. \tag{1}$$

Here, $\boldsymbol{u}, \boldsymbol{v}$, and $\boldsymbol{t}$ denote the attitude, velocity, and position, respectively. We assume that the initial state of the IMU, $\boldsymbol{x}_0$, is known and that the IMU collects three-dimensional (3D) acceleration and 3D angular velocity over time $\{\boldsymbol{a}_i, \boldsymbol{\omega}_i\}_{i=1}^n$.

Herein, we employ a rotation vector [9] to represent the 3D attitude $\boldsymbol{u}$. The definition of the rotation vector representation and the operations performed with it are available in Appendix.

Naive integration

The simplest method for IMU state estimation is the naive integration of the measured data. We estimate the attitude by integrating 3D angular velocity data obtained by the IMU. Assuming that the time step Δt is small, the attitude at each time step can be incrementally calculated as follows:

$$\boldsymbol{u}_{i+1} = \boldsymbol{u}_i * (\boldsymbol{\omega}_i \Delta t), \tag{2}$$

where the $*$ operator represents the concatenation of two rotation vectors.

Velocity can be estimated through the integration of the 3D acceleration data. Note that we need to compensate for the gravitational acceleration, which is denoted by $\boldsymbol{g}$, as follows:

$$\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + (\boldsymbol{R}_i \boldsymbol{a}_i - \boldsymbol{g}) \Delta t, \tag{3}$$

where $\boldsymbol{R}_i$ represents the rotation matrix representation of the attitude $\boldsymbol{u}_i$ and $\boldsymbol{R}_i \boldsymbol{a}_i$ represents the acceleration transformed into the world frame.

Position can be estimated through the integration of velocity data as follows:

$$\boldsymbol{t}_{i+1} = \boldsymbol{t}_i + \boldsymbol{v}_i \Delta t. \tag{4}$$

Sagawa et al.'s error correction method

The aforementioned integration typically leads to significant error accumulation. We now review a method proposed by Sagawa et al. [8] for correcting such errors.

This method assumes that the initial and final states are known, so the relative difference between the initial and final states is used as a constraint on trajectory reconstruction. The estimated time-series states are corrected by uniformly distributing the accumulated errors along the time sequence so that the error at the final state is canceled out.

This method uses the following steps:

1. Initial estimates of attitude are calculated via simple integration. The accumulated errors in these attitude values are corrected using the constraint.
2. Velocities are estimated via integration using the acceleration data and the corrected attitudes. The accumulated errors are again corrected using the constraint.
3. Positions are estimated through the integration of the corrected velocities, and these estimations are corrected with respect to the constraint.

Although the algorithm in the original paper [8] was described using the rotation matrix representation, we will detail the steps in rotation vector notation for easy comparison with the proposed algorithm.

Attitude correction

The attitudes estimated via simple integration are denoted as $\{\tilde{\boldsymbol{u}}_i\}_{i=1}^n$, and the known final attitude is denoted by $\boldsymbol{u}_n^{\mathrm{GT}}$. The accumulated attitude error can be expressed as a single rotation:

$$\boldsymbol{u}^{\mathrm{E}} = (-\tilde{\boldsymbol{u}}_n) * \boldsymbol{u}_n^{\mathrm{GT}}.$$

Attitude error correction is accomplished by uniformly distributing the error according to the time sequence as follows:

$$\hat{\boldsymbol{u}}_i = \tilde{\boldsymbol{u}}_i * (\frac{i}{n}\boldsymbol{u}^{\mathrm{E}}).$$

Velocity correction

The velocity error is corrected so that the estimated final velocity $\tilde{\boldsymbol{v}}_n$ matches the given final velocity $\boldsymbol{v}_n^{\mathrm{GT}}$. The accumulated error is assumed to be caused by a constant acceleration bias, $\boldsymbol{a}^{\mathrm{E}}$, which is calculated as follows:

$$\boldsymbol{a}^{\mathrm{E}} n\Delta t = \boldsymbol{v}_n^{\mathrm{GT}} - \tilde{\boldsymbol{v}}_n.$$

The corrected velocity sequence $\{\hat{\boldsymbol{v}}_i\}_{i=1}^n$ is then calculated as follows:

$$\hat{\boldsymbol{v}}_i = \tilde{\boldsymbol{v}}_i + \boldsymbol{a}^{\mathrm{E}} i\Delta t.$$

Position correction

Although the position correction step was not detailed in the original paper [8], we assume that Sagawa et al. repeated the above process. The difference between the final position estimated via simple integration $\tilde{\boldsymbol{t}}_n$ and the given final position $\boldsymbol{t}_n^{\mathrm{GT}}$ assumed to be caused by a constant velocity bias, $\boldsymbol{v}^E$, which satisfies the following relation:

$$\boldsymbol{v}^{\mathrm{E}} n\Delta t = \boldsymbol{t}_n^{\mathrm{GT}} - \tilde{\boldsymbol{t}}_n.$$

The corrected position sequence $\{\hat{\boldsymbol{t}}_i\}_{i=1}^n$ is then calculated as follows:

$$\hat{\boldsymbol{t}}_i = \tilde{\boldsymbol{t}}_i + \boldsymbol{v}^{\mathrm{E}} i\Delta t.$$

Proposed method

The aforementioned error correction employed a single constraint. Thus, we extend the problem to multiple constraints.

We treat the problem as an optimization of a graphical model shown in Fig. 1. The state at each time is modeled as a graph node. The constraints representing a relation between two nodes are modeled as graph edges.

Objective function

We define a vector containing all time-series state parameters as follows:

$$\boldsymbol{x} := [\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top]^\top.$$

This vector is used as an objective variable in our optimization problem.

Next, given a set of constraints, we aim to find the optimal vector $\boldsymbol{x}^*$. Constraint $\boldsymbol{c}_{ij}$ is a nine-dimensional vector representing the state of $\boldsymbol{x}_j$ relative to $\boldsymbol{x}_i$, which is obtained from IMU measurements and prior knowledge about the target motion. The problem is formulated as the maximization of the posterior density as follows:

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} p(\boldsymbol{x}|\{\boldsymbol{c}_{ij}\}_{\langle i,j\rangle\in C}). \tag{5}$$

Assuming that the errors in the constraints are statistically independent of each other and Gaussian, we can express the posterior density as follows:

$$p(\boldsymbol{x}|\{\boldsymbol{c}_{ij}\}_{\langle i,j\rangle\in C}) = \prod_{\langle i,j\rangle\in C} f(\boldsymbol{e}_{ij}(\boldsymbol{x})|\boldsymbol{0}, \boldsymbol{\Sigma}_{ij}), \tag{6}$$

where $f(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the probability density function of the normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\boldsymbol{e}_{ij}(\boldsymbol{x})$ denotes the error of the constraint $\boldsymbol{c}_{ij}$.

By taking the logarithm of Eq. (6), the optimization problem can be rewritten as follows:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} F(\boldsymbol{x}), \tag{7}$$

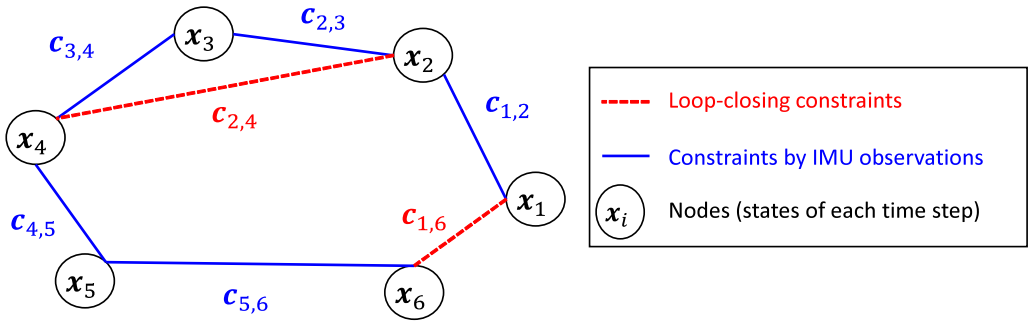

Fig. 1 Sample graphical model

$$F(\boldsymbol{x}) := \sum_{\langle i,j\rangle \in C} \boldsymbol{e}_{ij}(\boldsymbol{x})^\top \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij}(\boldsymbol{x}), \tag{8}$$

$$\boldsymbol{\Omega}_{ij} := \boldsymbol{\Sigma}_{ij}^{-1}. \tag{9}$$

Constraints

Each component (attitude, velocity, and position) in the constraint $\boldsymbol{c}_{ij}$ is denoted as $\boldsymbol{u}^{(ij)}$, $\boldsymbol{v}^{(ij)}$, and $\boldsymbol{t}^{(ij)}$, respectively. Two types of constraints are employed: a constraint between consecutive nodes ($j = i + 1$) and that between non-sequential nodes (loop-closing constraints). The former type of constraints can be calculated using IMU measurements as follows:

$$\boldsymbol{c}_{ij} = \begin{bmatrix} \boldsymbol{\omega}_i \Delta t \\ (\boldsymbol{R}_i \boldsymbol{a}_i - \boldsymbol{g}) \Delta t \\ \boldsymbol{v}_i \Delta t \end{bmatrix}.$$

In contrast, the non-sequential constraints can be obtained from prior knowledge about the motion or observations with external sensors. The single constraint used in Sagawa et al.'s method is given in this form. For example, a constraint that the initial (time $t = 0$) and final (time $t = n$) states are identical can be expressed as follows:

$$\boldsymbol{c}_{0,n} = [0, 0, 0, 0, 0, 0]^\top.$$

Error function

We define the error function as follows:

$$\boldsymbol{e}_{ij}(\boldsymbol{x}) := \begin{bmatrix} (-\boldsymbol{u}^{(ij)}) * (-\boldsymbol{u}_i) * (\boldsymbol{u}_j) \\ -\boldsymbol{v}^{(ij)} + \boldsymbol{v}_j - \boldsymbol{v}_i \\ -\boldsymbol{t}^{(ij)} + \boldsymbol{t}_j - \boldsymbol{t}_i \end{bmatrix}.$$

The error function returns a zero vector when the state of node j relative to node i matches $\boldsymbol{c}_{ij}$.

Optimization

We employ the Gauss–Newton method to minimize function F in Eq. (8). This method is a gradient descent method that can be used for nonlinear least-squares problems. We initiate the objective variable with an initial estimate, $\hat{\boldsymbol{x}}$, and iterate over the following three steps until convergence:

1. Linearization: Calculate $\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d})$, i.e., the linear approximation of $F(\boldsymbol{x})$ at $\boldsymbol{x} = \hat{\boldsymbol{x}}$.
2. Minimization: Determine the update vector $\boldsymbol{d}$, which minimizes $\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d})$.
3. Update: Update the current estimate $\hat{\boldsymbol{x}}$ using $\boldsymbol{d}$.

Linearization

In the first step, we obtain the linearized objective function $\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d})$ as follows:

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

Here, $\boldsymbol{J}_{ij}$ is the Jacobian matrix of $\boldsymbol{e}_{ij}(\boldsymbol{x})$ at $\boldsymbol{x} = \hat{\boldsymbol{x}}$.

The linearized objective function $\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d})$ can be expanded into a quadratic function as follows:

$$\bar{F}_{\hat{\boldsymbol{x}}}(\boldsymbol{d}) = \boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d} + 2\boldsymbol{b}^\top \boldsymbol{d} + \text{const.}, \tag{10}$$

where

$$\begin{aligned} \boldsymbol{H} &:= \sum_{\langle i,j\rangle \in C} \boldsymbol{J}_{ij}^\top \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}, \\ \boldsymbol{b} &:= \sum_{\langle i,j\rangle \in C} \boldsymbol{e}_{ij}(\hat{\boldsymbol{x}})^\top \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}. \end{aligned} \tag{11}$$

Minimization

By setting the derivative of Eq. (10) to zero, the minimizer $\hat{\boldsymbol{d}}$ is obtained by solving the following linear equation:

$$\boldsymbol{H}\boldsymbol{d} = -\boldsymbol{b}. \tag{12}$$

$\boldsymbol{H}$ is a $9n \times 9n$ matrix, and it grows rapidly as the number of nodes increases. Therefore, naive methods, including Gauss–Seidel minimization, require a large amount of time for computations. Since $\boldsymbol{H}$ is symmetric, positive-definite, and sparse (i.e., the number of loop-closing constraints is small), the linear system can be efficiently solved via sparse Cholesky decomposition or the preconditioned conjugate gradient method [10, 11].

Update

Finally, using the obtained minimizer $\hat{\boldsymbol{d}}$, we update the current estimates using the following equation:

$$\hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} + \hat{\boldsymbol{d}}. \tag{13}$$

Relevance to graph-based simultaneous localization and mapping

The formulation can be considered as an extension of graph-based simultaneous localization and mapping (SLAM) [12, 13]. Graph-based SLAM produces attitude and position estimates, and the proposed method extends the state vector with velocity as an extra degree of freedom.

Improving initial estimates

The computational cost of the proposed method is higher than those of the graph-based SLAM methods for the same number of nodes and constraints as $\boldsymbol{H}$ becomes larger. Therefore, to achieve practical processing time,

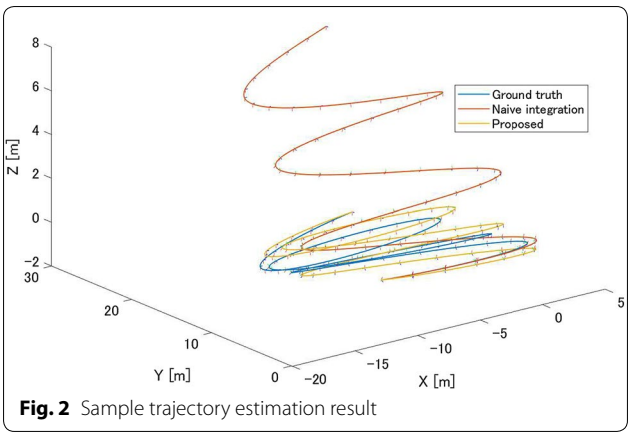

Fig. 2 Sample trajectory estimation result

we employ an initialization technique, which is described below.

The positions estimated via simple integration are typically far from the optimum value. If we begin optimization with such poor initial estimates, the optimization will require a large amount of time to converge.

Therefore, we improve the initial estimates by solving a sub-problem. The sub-problem employs the same graph-based formulation as the main problem and only optimize the attitude and velocity estimates. More specifically,

$$\boldsymbol{x}' := \left[\boldsymbol{u}_0^\top, \boldsymbol{v}_0^\top, \boldsymbol{u}_1^\top, \boldsymbol{v}_1^\top, \ldots, \boldsymbol{u}_n^\top, \boldsymbol{v}_n^\top\right]^\top$$

is used as an objective variable. The attitude and velocity components of the constraints were also extracted and employed.

We initialize the position estimates for the main problem by integrating the attitudes and velocities returned by sub-problem optimization, thereby reducing the iterations required for convergence in the main optimization problem.

Results

Simulation experiments

Experiments with simulated data were conducted as follows. IMUSim [14] was employed to generate 100 different random trajectories with ideal 100-Hz IMU measurements. The duration of each generated trajectory was 3 s.

To improve the realism of the simulation, artificial white noise was added to the ideal measurements. The standard deviations of the noise terms were 0.1 m/s^2 for acceleration and 0.3°/s for angular velocity. Bias errors of 0.2°/s were also added to the angular velocity data.

First, we tested the processing methods for motions with only one constraint. The estimation accuracy of the proposed method was compared with those of the naive integration and Sagawa et al.'s conventional method [8].

The examples of estimated trajectories and RMS errors of each method are shown in Fig. 2 and summarized in Fig. 3. Regarding position and velocity estimations, the proposed method returned errors that were almost same as those returned by the other methods, which makes sense because both methods used the same information.

However, the proposed method exhibited significantly smaller attitude errors in comparison with Sagawa et al.'s method ($p < 0.001$ using the Wilcoxon signed-rank test). The typical attitude estimation results are plotted in Fig. 4, which is generated using the same motion as shown in Fig. 2. This plot shows that both correction

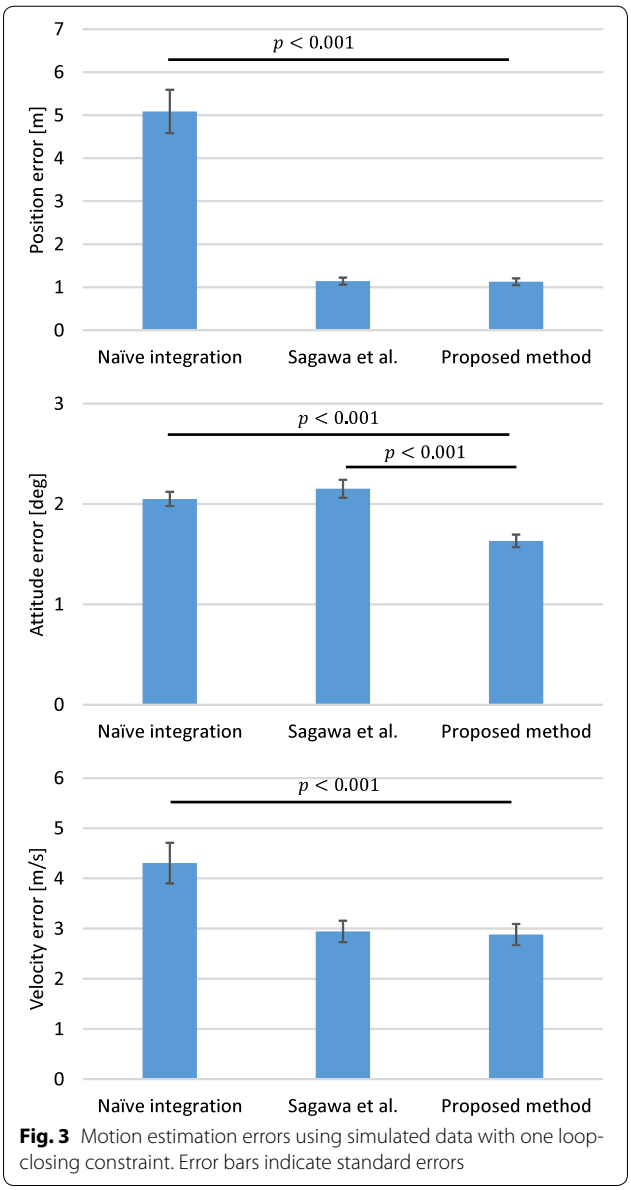

Fig. 3 Motion estimation errors using simulated data with one loop-closing constraint. Error bars indicate standard errors

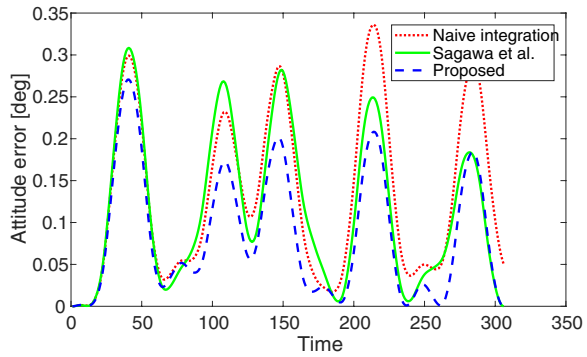

Fig. 4 Attitude estimation errors for comparison

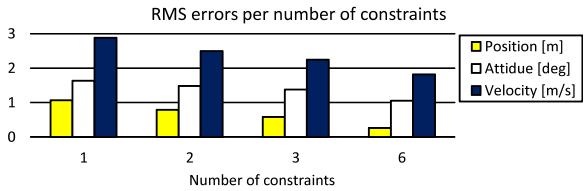

Fig. 5 RMS errors with different number of constraints (simulated data)

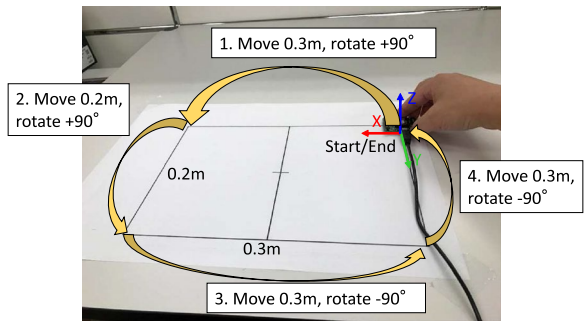

Fig. 6 IMU data collection procedure of trajectory estimation experiment

methods accurately estimated the final attitude; however, Sagawa et al.'s method exhibited larger errors for intermediate states. This was because the method uniformly distributed the attitude errors along a single rotation axis. Thus, the correction was not expected to necessarily match with the direction of the errors.

Next, we tested the method with multiple constraints. Several observation points were chosen in the simulated data sequences, and the true IMU states at these points were used as additional constraints. The relation between the number of constraints used and the estimation accuracy is summarized in Fig. 5. This figure clearly shows that the estimation accuracy improves as the number of constraints increases.

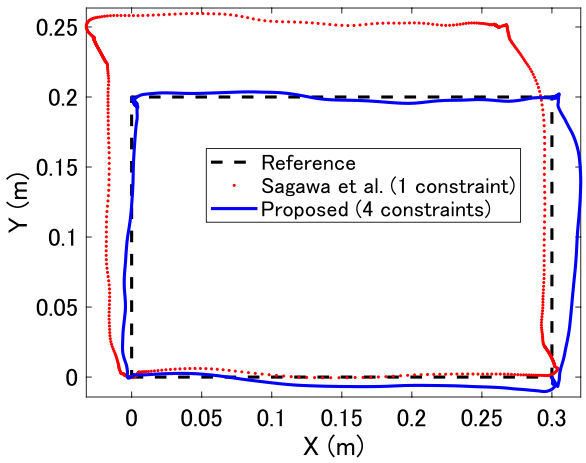

Fig. 7 Two dimensional projection of the estimated trajectories using real IMU data

Trajectory estimation using real IMU data

The accuracy of trajectory estimation was evaluated using real IMU data collected with an Xsense MTi-3 IMU at 100 Hz. The motion measured is shown in Fig. 6. The IMU was moved to four vertices of a 0.3 m × 0.2 m rectangle and returned to the initial position and orientation. The motion lasted approximately 7 s. To perform the calibration of the IMU's acceleration and angular velocity bias, approximately 3.0 s of motionless time was included at the beginning of the motion.

We again compared the proposed method and Sagawa et al.'s method [8] in terms of estimation accuracy. Both methods were provided the same constraint, i.e., the initial and final states were identical. Extra constraints were employed in the proposed method, which are defined by the relative positions of three vertices and the assumption that the device stops moving at these vertices. The times when the IMU visited the vertices was calculated based on the acceleration spikes caused by collisions with the table surface.

The estimated 3D trajectories were projected into a two dimensional plane (Fig. 7) for comparison with the reference trajectory. The Fréchet distance (a similarity measure) between the estimated trajectory and reference trajectory was employed for measuring errors. The errors were 0.062 m from Sagawa et al.'s method and 0.022 m from the proposed method.

The changes in the estimation accuracy for different number of constraints are plotted in Fig. 8. This figure shows that the estimation accuracy improves as the number of constraints increases.

The processing times for the proposed method and Sagawa et al.'s method were 0.58 and 0.022 s, respectively, on a laptop equipped with a Core i7-7600U processor

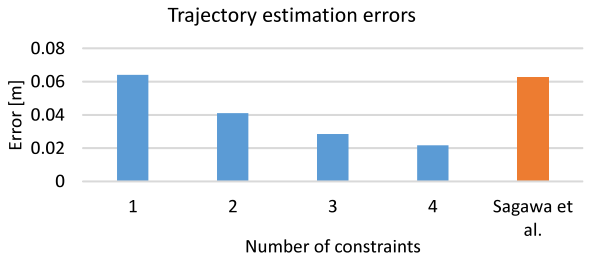

Fig. 8 Trajectory estimation errors with different number of constraints (real IMU data)

(mean of 10 trials). Although the proposed method performed slower than Sagawa et al.'s method, the performance of our method does not appear to be a significant problem for the purpose of offline motion analysis.

Attitude estimation using real IMU data

The attitude estimation accuracy was also evaluated using the MTi-3 IMU. In this evaluation, the IMU was rotated randomly and returned to the initial state, as depicted in Fig. 9. The constraint that the initial and final states are identical was employed. The mean attitude errors for the collected 53 sequences of the IMU data are shown in Fig. 10. As can be seen from the figure, the estimation error of the proposed method was smaller than that of Sagawa et al.'s method ($p = 0.0376$).

Analysis of the effect of data frequency and length

To evaluate how the data acquisition rate affects the performance of the proposed method, we conducted an additional experiment using simulated IMU data of 1000 Hz from the same 100 random motions employed in the abovementioned simulation experiments. The estimation results are summarized in Fig. 11. The graph shows similar trend with that of 100 Hz (Fig. 3). The mean processing time of the proposed method was 3.4 s for 1000 Hz data and it is 15 times slower than that for 100 Hz (0.23 s).

The estimation accuracy for longer motions was evaluated using the same settings as the trajectory estimation experiments using the MTi-3 IMU. Two additional data sequences were collected by slowly moving along the rectangle (same procedure as shown in Fig. 6); the duration of the data was approximately 14 and 21 s. Trajectory estimation results are summarized in Fig. 12. It was observed that the estimation accuracy of both the proposed method and Sagawa et al.'s method degraded rapidly as motion length increased.

Conclusions and future work

Using IMU data, the proposed motion estimation method corrects the accumulated errors via least-squares fitting with constraints that are obtained from IMU measurements and from prior knowledge of the motion. This method can incorporate more than one constraint in error correction steps, thereby improving the estimation accuracy. The effect of accuracy improvement was

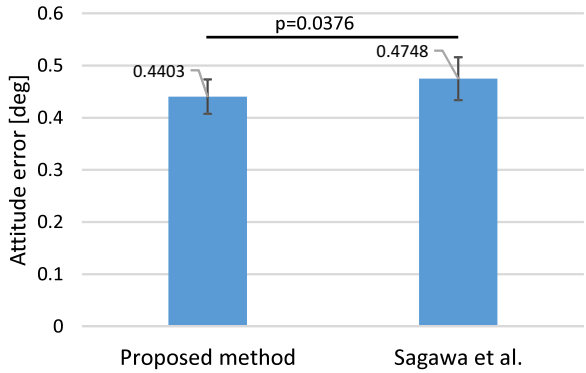

Fig. 10 Results of attitude estimation experiments using real IMU data. Error bars indicate standard errors

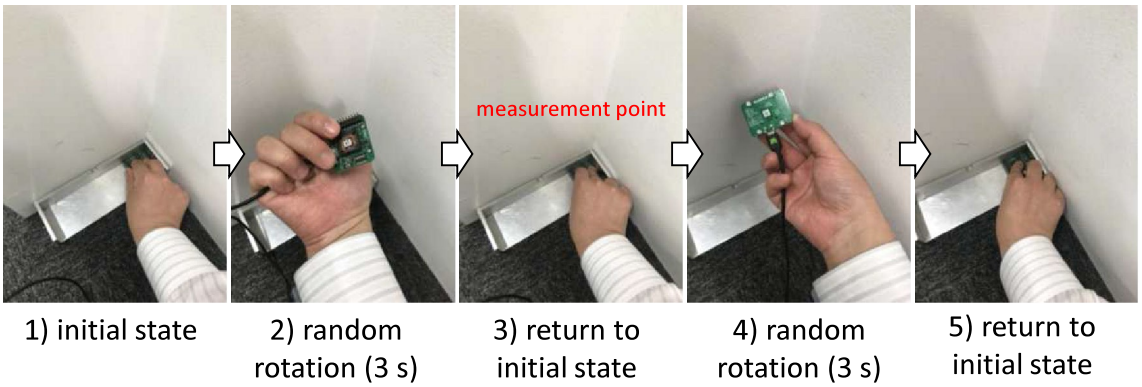

Fig. 9 IMU data collection procedure of attitude estimation experiment

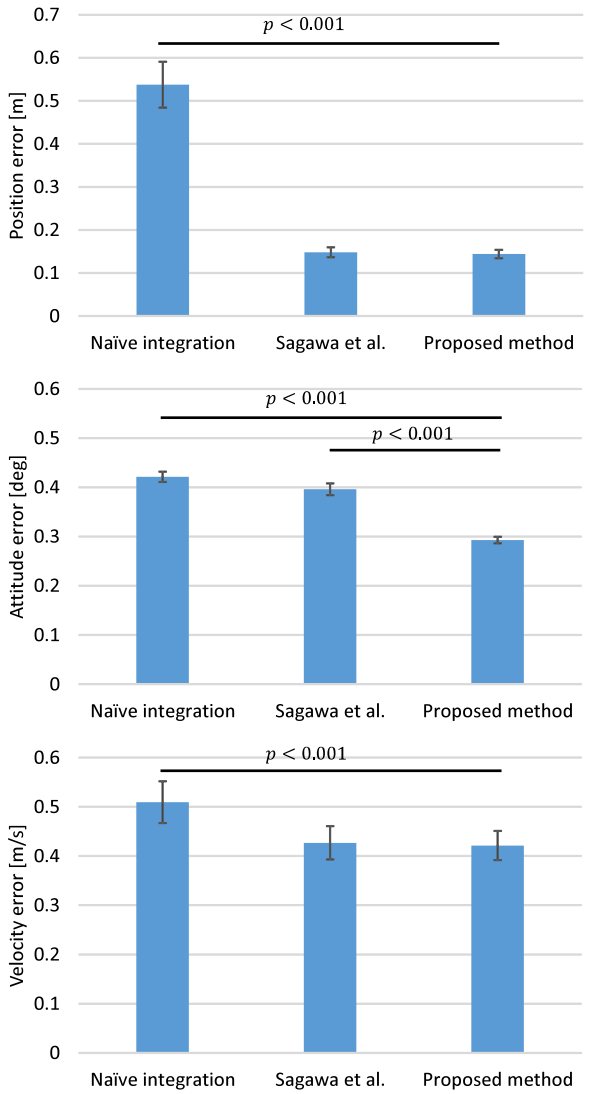

Fig. 11 Estimation errors using 1000 Hz simulated data

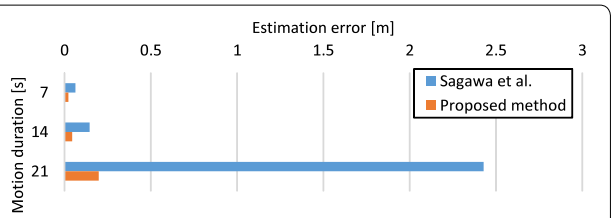

Fig. 12 Trajectory estimation results for long motions (real IMU data)

verified by experiments using simulated and real data. When using a single constraint, the proposed method estimated attitude more accurately than the conventional methods. The computation time of the proposed method was practical for motions with duration of several seconds, thanks to the use of an initialization technique. Nevertheless, the computational cost will be large on data with long duration or high frequency. One way to alleviate the problem would be node thinning.

In future work, we plan to apply this method to actual sports data. An analysis of table tennis rallies is in progress.

The proposed method has a limitation: it assumes that the constraints are independent; however, biased measurements can violate this assumption. A mathematical model that considers gyroscope and accelerometer biases can compensate for this limitation.

Authors' contributions

The author read and approved the final manuscript.

Competing interests

The author declares that he has no competing interests.

Ethics approval and consent to participate

Not applicable.

Appendix

Rotation vector definition

A rotation vector is a 3D vector representing a 3D rotation. It can be calculated from a 3D unit vector representing the rotation axis $\boldsymbol{a} = [a_x, a_y, a_z]^\top$ and the angle of rotation θ[rad] as follows:

$$\boldsymbol{v} := \theta \boldsymbol{a}. \tag{14}$$

The inverse rotation can be calculated as follows:

$$\boldsymbol{v}^{-1} = -\boldsymbol{v}. \tag{15}$$

For brevity, the norm of $\boldsymbol{v}$ is denoted as v.

Conversion between unit quaternion

Conversion from a rotation vector to unit quaternion $\boldsymbol{q} = [q_w, q_x, q_y, q_z]^\top$ can be expressed as follows:

$$\boldsymbol{q}_v(\boldsymbol{v}) := \begin{bmatrix} \cos\frac{v}{2} \\ \frac{v_1}{v}\sin\frac{v}{2} \\ \frac{v_2}{v}\sin\frac{v}{2} \\ \frac{v_3}{v}\sin\frac{v}{2} \end{bmatrix}. \tag{16}$$

Conversion from unit quaternion to rotation vector can be expressed as follows:

$$\boldsymbol{v}_q(\boldsymbol{q}) := \frac{2\arccos(q_w)}{\sqrt{1-q_w^2}} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}. \tag{17}$$

Rotation vector to rotation matrix

A rotation vector can be converted to a rotation matrix as follows (*Rodrigues's formula*):

$$\boldsymbol{R} = \boldsymbol{I}_3 + \sin v \boldsymbol{K} + (1 - \cos v)\boldsymbol{K}^2, \tag{18}$$

$$\boldsymbol{K} := \frac{1}{\nu}\begin{bmatrix} 0 & -\nu_3 & \nu_2 \\ \nu_3 & 0 & -\nu_1 \\ -\nu_2 & \nu_1 & 0 \end{bmatrix}. \tag{19}$$

Rotation vector multiplication

The product of the two rotation vectors $\boldsymbol{\nu}$ and $\boldsymbol{u}$ is represented by $\boldsymbol{\nu} * \boldsymbol{u}$, which is calculated in terms of the quaternion product.

$$\boldsymbol{\nu} * \boldsymbol{u} := \boldsymbol{\nu}_q(\boldsymbol{q}_\nu(\boldsymbol{\nu})\boldsymbol{q}_\nu(\boldsymbol{u})) \tag{20}$$

This operation represents the concatenation of two rotations (first rotation by $\boldsymbol{\nu}$, and the next rotation by $\boldsymbol{u}$).

Derivatives of rotation vector multiplication

Derivatives of three rotation vector multiplication are calculated as follows:

$$\begin{aligned} \frac{\partial(\boldsymbol{u} * \boldsymbol{\nu} * \boldsymbol{w})}{\partial \boldsymbol{\nu}} &= H(\boldsymbol{q}_\nu(\boldsymbol{u} * \boldsymbol{\nu} * \boldsymbol{w}))Q(\boldsymbol{q}^{\boldsymbol{u}})\bar{Q}(\boldsymbol{q}^{\boldsymbol{w}})G(\boldsymbol{\nu}) \\ &:= U_2(\boldsymbol{u}, \boldsymbol{\nu}, \boldsymbol{w}), \end{aligned} \tag{21}$$

$$\begin{aligned} \frac{\partial(\boldsymbol{u} * \boldsymbol{\nu} * \boldsymbol{w})}{\partial \boldsymbol{w}} &= H(\boldsymbol{q}_\nu(\boldsymbol{u} * \boldsymbol{\nu} * \boldsymbol{w}))Q(\boldsymbol{q}^{\boldsymbol{u}})Q(\boldsymbol{q}^{\boldsymbol{\nu}})G(\boldsymbol{w}) \\ &:= U_3(\boldsymbol{u}, \boldsymbol{\nu}, \boldsymbol{w}). \end{aligned} \tag{22}$$

Here, H, G, Q, and $\bar{Q}$ are given as follows [9]:

$$\begin{aligned} H(\boldsymbol{q}) &:= \frac{\partial \boldsymbol{\nu}_q(\boldsymbol{q})}{\partial \boldsymbol{q}} \\ &= \begin{bmatrix} 2cq_x(dq_w - 1) & 2d & 0 & 0 \\ 2cq_y(dq_w - 1) & 0 & 2d & 0 \\ 2cq_z(dq_w - 1) & 0 & 0 & 2d \end{bmatrix}, \end{aligned} \tag{23}$$

where $c := \frac{1}{1-q_w^2}$, $d := \frac{\arccos(q_w)}{\sqrt{1-q_w^2}}$,

$$\begin{aligned} G(\boldsymbol{\nu}) &:= \frac{\partial \boldsymbol{q}_\nu(\boldsymbol{\nu})}{\partial \boldsymbol{\nu}} \\ &= \frac{S}{2\nu}\begin{bmatrix} -\nu_1 & -\nu_2 & -\nu_3 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \\ &\quad + \frac{a}{2\nu^3}\begin{bmatrix} 0 & 0 & 0 \\ \nu_1^2 & \nu_1\nu_2 & \nu_1\nu_3 \\ \nu_1\nu_2 & \nu_2^2 & \nu_2\nu_3 \\ \nu_1\nu_3 & \nu_2\nu_3 & \nu_3^2 \end{bmatrix}, \end{aligned} \tag{24}$$

where $S := \sin\frac{\nu}{2}$, $a := \nu\cos\frac{\nu}{2} - 2S$,

$$Q(\boldsymbol{q}) := \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}, \tag{25}$$

$$\bar{Q}(\boldsymbol{q}) := \begin{bmatrix} q_w & -q_x & -q_y & -q_z \\ q_x & q_w & q_z & -q_y \\ q_y & -q_z & q_w & q_x \\ q_z & q_y & -q_x & q_w \end{bmatrix}. \tag{26}$$

Jacobian matrix calculation

The Jacobian matrix of $\boldsymbol{e}_{ij}(\boldsymbol{x})$ at $\boldsymbol{x} = \hat{\boldsymbol{x}}$ has the following sparse structure:

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

Here, $\boldsymbol{J}_{ij}^{(i)}$ and $\boldsymbol{J}_{ij}^{(j)}$ are the partial derivatives of $\boldsymbol{e}_{ij}(\boldsymbol{x})$ by $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively, and they can be calculated as:

$$\begin{aligned} \boldsymbol{J}_{ij}^{(j)} &:= \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x})}{\partial \boldsymbol{x}_j}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \\ &= \frac{\partial}{\partial \boldsymbol{x}_j}\begin{bmatrix} (-\boldsymbol{u}^{(ij)}) * (-\boldsymbol{u}_i) * (\boldsymbol{u}_j) \\ -\boldsymbol{\nu}^{(ij)} + \boldsymbol{\nu}_j - \boldsymbol{\nu}_i \\ -\boldsymbol{t}^{(ij)} + \boldsymbol{t}_j - \boldsymbol{t}_i \end{bmatrix}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \\ &= \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{x}_j}(-\boldsymbol{u}^{(ij)}) * (-\boldsymbol{u}_i) * (\boldsymbol{u}_j)\big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_3 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 \end{bmatrix} \\ &= \begin{bmatrix} U_3(-\boldsymbol{u}^{(ij)}, -\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_j) & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_3 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 \end{bmatrix}, \end{aligned} \tag{27}$$

$$\begin{aligned} \boldsymbol{J}_{ij}^{(i)} &:= \frac{\partial \boldsymbol{e}_{ij}(\boldsymbol{x})}{\partial \boldsymbol{x}_i}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \\ &= \frac{\partial}{\partial \boldsymbol{x}_i}\begin{bmatrix} (-\boldsymbol{u}^{(ij)}) * (-\boldsymbol{u}_i) * (\boldsymbol{u}_j) \\ -\boldsymbol{\nu}^{(ij)} + \boldsymbol{\nu}_j - \boldsymbol{\nu}_i \\ -\boldsymbol{t}^{(ij)} + \boldsymbol{t}_j - \boldsymbol{t}_i \end{bmatrix}\Big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} \\ &= \begin{bmatrix} \frac{\partial}{\partial \boldsymbol{x}_i}(-\boldsymbol{u}^{(ij)}) * (-\boldsymbol{u}_i) * (\boldsymbol{u}_j)\big|_{\boldsymbol{x}=\hat{\boldsymbol{x}}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & -\boldsymbol{I}_3 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & -\boldsymbol{I}_3 \end{bmatrix} \\ &= -\begin{bmatrix} U_2(-\boldsymbol{u}^{(ij)}, -\hat{\boldsymbol{u}}_i, \hat{\boldsymbol{u}}_j) & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_3 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I}_3 \end{bmatrix}. \end{aligned} \tag{28}$$

Publisher's Note

Received: 31 January 2018 Accepted: 16 May 2018
Published online: 29 May 2018

References

1. Seaman A, McPhee J (2012) Comparison of optical and inertial tracking of full golf swings. Procedia Eng 34:461–466
2. TruSwing™. http://www.garmin.com.hk/products/intosports/truswing/

3. Smart Tennis Sensor for Tennis Rackets. https://www.sony.com/electronics/smart-devices/sse-tn1w
4. Woodman OJ (2007) An introduction to inertial navigation. Technical report, University of Cambridge
5. Madgwick SOH, Harrison AJL, Vaidyanathan R (2011) Estimation of IMU and MARG orientation using a gradient descent algorithm. In: IEEE international conference on rehabilitation robotics, pp 1–7
6. Ojeda L, Borenstein J (2007) Non-GPS navigation for security personnel and first responders. J Navig 60(3):391–407
7. Sagawa K, Ohkubo K (2015) 2D trajectory estimation during free walking using a tiptoe-mounted inertial sensor. J Biomech 48(10):2054–2059
8. Sagawa K, Abo S, Tsukamoto T, Kondo I (2009) Forearm trajectory measurement during pitching motion using an elbow-mounted sensor. J Adv Mech Design Syst Manuf 3(4):299–311
9. Diebel J (2006) Representing attitude: Euler angles, unit quaternions, and rotation vectors. Report, Stanford University
10. Konolige K, Grisetti G, Kummerle R, Burgard W, Limketkai B, Vincent R (2010) Efficient sparse pose adjustment for 2D mapping. In: Proceedings of the IEEE international conference on intelligent robots & systems (IROS), pp 22–9
11. Montemerlo M, Thrun S (2006) Large-scale robotic 3-D mapping of urban structures. In: Ang MH, Khatib O (eds) Experimental robotics IX: the 9th international symposium on experimental robotics, vol 21. Springer, Berlin, pp 141–150
12. Lu F, Milios E (1997) Globally consistent range scan alignment for environment mapping. Auton Robots 4:333–349
13. Grisetti G, Kummerle R, Stachniss C, Burgard W (2010) A tutorial on graph-based slam. IEEE Intell Transp Syst Mag 2(4):31–43
14. Young AD, Ling MJ, Arvind DK (2011) IMUSim: a simulation environment for inertial sensing algorithm design and evaluation. In: Proceedings of international conference on information processing in sensor networks (IPSN), pp 199–210