NATIONAL UNIVERSITY OF SINGAPORE

Numerical Integration for Rotational Kinematics

Term Paper for ME5701:
Mathematics for Engineering Research

Wang Bingheng
NUSSTU ID: A0226338A
wangbingheng@u.nus.edu
Instructors:
Prof.Chirikjian and Prof.Mengaldo

16 Nov. 2021

Abstract

This term paper elaborates on the numerical integration approaches for rotational kinematics written in the form of matrix ordinary differential equation (ODE). The rotation matrix involved in the rotational kinematics evolves on a manifold and pertains to a special orthogonal group. It is desired to develop algorithms that can integrate the matrix ODE numerically while preserving these properties of rotation matrices. Four numerical integration algorithms are discussed in this report, forward Euler method, Runge-Kutta method, QR-decomposition-based method, and Lie group integrator method. The formulations of these methods are first presented in detail. Then, the ability to preserve the matrix structure of these methods is analyzed by using a Taylor series of the matrix exponential. Finally, the performances of these methods are compared with each other in numerical simulation with real flight data. It is shown that the Lie group integrator outperforms the others in terms of the structure-reserving ability and the numerical accuracy relative to the ground truth rotation matrix. It is also shown that with the order increasing, the high-order Runge-Kutta method is able to approximate the Lie group integrator well.

List of Contents

1 Introduction

In lecture 6, it was discussed by Prof.Chirikjian that numerically integrating the following rotational kinematics,

$$\frac{d}{dt}\boldsymbol{R}(t) = \boldsymbol{\omega}_L^{\wedge}(t)\,\boldsymbol{R}(t) \tag{1}$$

where $\boldsymbol{\omega}_L^{\wedge}$ is a skew-symmetric matrix corresponding to the angular velocity $\boldsymbol{\omega}_L$, might cause the matrix $\boldsymbol{R}(t) \in \mathbb{R}^{3\times 3}$ not to be a rotation matrix any longer. To become a rotation matrix, the $\boldsymbol{R}(t)$ in the above matrix ordinary differential equation (ODE) should be confined to a Lie group (i.e., special orthogonal group in three dimensions, $SO(3)$) and be characterized by two matrix structures, $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_{3\times 3}$ and $\det(\boldsymbol{R}) = 1$. It is favorable to preserve these matrix structures while integrating the matrix ODE (1) numerically, since rotation matrices are prevalent in robotics and play crucial roles in motion planning and control, such as model predictive control for trajectory optimization and tracking [1]. This project aims to investigate which numerical integration algorithms are able to preserve the matrix structures of rotation matrices and how they work.

Numerical integration for rotational kinematics has attracted considerable attention in both mathematics and engineering fields, and numerous algorithms have been proposed. The most straightforward approach is the forward Euler integration which discretizes the kinematics (1) using the first-order divided difference and add the finite difference at the current sampling time to the $\boldsymbol{R}(t)$ by matrix addition. The accuracy of this method is very vulnerable to the discrete step, with the step increasing, the accuracy degenerates dramatically. The Runge-Kutta method improves the Euler method to have greater accuracy while reducing the dependence on the step. In each step, unlike the finite difference of the forward Euler method that is evaluated at the beginning of the step, the Runge-Kutta method calculates the finite difference by a weighted summation of many derivatives evaluated at different points during the step [2]. Despite the improvements, the Runge-Kutta method still uses matrix addition to update the $\boldsymbol{R}(t)$, which is essentially a translation of the $\boldsymbol{R}(t)$ not a rotation. Therefore, these classical methods based on matrix addition fail to preserve the desired matrix structures [3].

The failures of the classical methods have motivated the development of Lie group integrator methods. Given the fact that the elements in a Lie group evolve on the manifolds, the Lie group integrators are able to ensure the numerical solution of the matrix ODE (1) evolves on the same manifold as the analytical solution. This ability of the Lie group integrators was first demonstrated in the pioneering work [4], which proposed two kinds of single-step and multi-step numerical integration algorithms using the vector flows with 'frozen' coefficients. Specifically, the authors in [4] showed that the solution of the matrix ODE (1) with the 'frozen' $\boldsymbol{\omega}_L^{\wedge}$ evolves on the same manifold as the analytical solution, and this enables the proposed algorithm to obtain the numerical solution accurately. On the basis of this idea, Munthe-Kaas [5] generalized the classical 4th-order Runge-Kutta method to differential equations on Lie groups by modifying the 'frozen' coefficients and replacing the matrix addition used in the classical methods with the matrix exponential map. The author further proposed a family of Runge-Kutta type methods of arbitrarily high order for differential equations on Lie groups using the Lie algebras acting on the manifolds [6]. The history, theory, and associated applications of the Lie group integrators are excellently summarized in the two review papers [7; 8].

Most recently, unlike the above Lie group integrators, Luc Jaulin [9] proposed a re-normalization method which projects the numerical solution of the matrix ODE (1) obtained by the classical methods onto $SO(3)$ using QR factorization. Although this method is able to preserve the matrix structures of rotation matrices, its effectiveness relies heavily on the accuracy of the classical numerical methods that compute the $\boldsymbol{R}(t)$ before the re-normalization. To have more insights, this paper compares the aforementioned four numerical methods with each other comprehensively using a real flight data-set, which was collected by making a quadrotor unmanned aerial vehicle perform the agile flights in one of the world's largest indoor motion capture system [10].

The rest of this paper is organized as follows. Section 2 defines the preliminaries used in the report. Section 3 details the derivations of the forward Euler method, the 4th-order Runge-Kutta method, the QR-factorization-based

method, and the Lie group integrator method. Section 4 gives the theoretical comparisons between these methods and highlights the structure-preserving feature of the Lie group integrator. Section 5 presents the numerical comparisons in simulation. Section 6 concludes the report.

2 Preliminaries

To make the report concise and readable, some important notations and concepts are defined in this section. Scalars are denoted in non-bold such as a and A, vectors in lowercase bold $\boldsymbol{a}$, and matrices in uppercase bold $\boldsymbol{A}$. The subscript L denotes a vector as seen in world frame $\mathcal{W}$ and the subscript R denotes a vector as seen in body frame $\mathcal{B}$. The operator $(\cdot)^\wedge$ defines the *hat* map that transfers a vector $\boldsymbol{a} = [a_1, a_2, a_3]^T \in \mathbb{R}^3$ to the associated skew-symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{3\times3}$, and the vector $\boldsymbol{a}$ is called the *dual vector* of $\boldsymbol{A}$ [11].

$$\boldsymbol{a}^\wedge = \boldsymbol{A} \doteq \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{2}$$

Accordingly, the counterpart is the *vee* map $(\cdot)^\vee$ that transfers $\boldsymbol{A}$ back to its dual vector $\boldsymbol{a}$, as shown below.

$$\boldsymbol{A}^\vee = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}^\vee \doteq [a_1, a_2, a_3]^T = \boldsymbol{a} \tag{3}$$

Any skew-symmetric matrix is equal to the negative transpose of itself, and this forms a set of matrices denoted as $\mathfrak{so}(3)$, i.e., $\boldsymbol{A} \in \mathfrak{so}(3) \doteq \left\{ \boldsymbol{A} \in \mathbb{R}^{3\times3} \,\middle|\, \boldsymbol{A} = -\boldsymbol{A}^T \right\}$. The skew-symmetric matrices have the following operation feature, namely: that the matrix multiplication of a skew-symmetric matrix $\boldsymbol{A}$ with an arbitrary vector $\boldsymbol{x}$ is the same as the cross product of the dual vector $\boldsymbol{a}$ with the same arbitrary vector [11], $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{a} \times \boldsymbol{x}$.

In robotics, it is a common practice to express the rotational kinematics (1) in terms of the angular velocity as seen in body frame, since the angular velocity $\boldsymbol{\omega}_R$ can be measured directly by an inertial measurement unit (IMU) mounted on a robot. Given the fact that any rotation matrix must satisfy $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_{3\times3}$, differentiating this matrix structure with respect to time yields

$$\dot{\boldsymbol{R}}\boldsymbol{R}^T + \boldsymbol{R}\dot{\boldsymbol{R}}^T = \frac{d}{dt}\left(\mathbb{I}_{3\times3}\right) = \boldsymbol{0}_{3\times3} \tag{4}$$

and this means that $\dot{\boldsymbol{R}}\boldsymbol{R}^T$ is a skew-symmetric matrix, since $\dot{\boldsymbol{R}}\boldsymbol{R}^T = -\boldsymbol{R}\dot{\boldsymbol{R}}^T = -\left(\dot{\boldsymbol{R}}\boldsymbol{R}^T\right)^T$. The rotation matrix $\boldsymbol{R}$ is defined to transform a vector $\boldsymbol{x}_R$ in $\mathcal{B}$ to its counterpart $\boldsymbol{x}_L$ in $\mathcal{W}$ by $\boldsymbol{x}_L = \boldsymbol{R}\boldsymbol{x}_R$. If $\boldsymbol{x}_R$ is a fixed position vector in $\mathcal{B}$, the velocity $\boldsymbol{v}_L$ can be obtained as $\boldsymbol{v}_L = \dot{\boldsymbol{R}}\boldsymbol{x}_R = \dot{\boldsymbol{R}}\boldsymbol{R}^T\boldsymbol{x}_L$. Comparing it with the velocity formula widely used in engineering $\boldsymbol{v}_L = \boldsymbol{\omega}_L \times \boldsymbol{x}_L$ leads to the equation

$$\boldsymbol{\omega}_L^\wedge = \dot{\boldsymbol{R}}\boldsymbol{R}^T \tag{5}$$

which holds true due to the skew-symmetry of $\dot{\boldsymbol{R}}\boldsymbol{R}^T$. It is evident that the rotational kinematics (1) is derived directly from the above equation. Using the relation $\boldsymbol{\omega}_L = \boldsymbol{R}\boldsymbol{\omega}_R$ and the equation $\boldsymbol{R}\boldsymbol{X}\boldsymbol{R}^T = (\boldsymbol{R}\boldsymbol{x})^\wedge$ where $\boldsymbol{X} = \boldsymbol{x}^\wedge$ (see the proof in Appendix 8), we can rewrite Eq.(5) as

$$\boldsymbol{\omega}_L^\wedge = \dot{\boldsymbol{R}}\boldsymbol{R}^T = (\boldsymbol{R}\boldsymbol{\omega}_R)^\wedge = \boldsymbol{R}\boldsymbol{\omega}_R^\wedge\boldsymbol{R}^T. \tag{6}$$

Therefore, the rotational kinematics in terms of $\boldsymbol{\omega}_R$ can be derived from Eq.(6) as

$$\frac{d}{dt}\boldsymbol{R}(t) = \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge(t) \tag{7}$$

which is used throughout this report for numerical integration.

3 Numerical Integration Algorithms

This section presents the detailed derivations of the four numerical integration algorithms of interest. For the forward Euler method and the 4th-order Runge-Kutta method, the rotational kinematics (7) can be written in the following ODE with a function $\boldsymbol{F}$.

$$\dot{\boldsymbol{R}}(t) = \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^{\wedge}(t) = \boldsymbol{F}(\boldsymbol{R}, \boldsymbol{\omega}_R) \tag{8}$$

3.1 Forward Euler Method

The Euler method is widely used in numerical integration thanks to its simple concept and easy implementation. The idea of Euler method is to approximate a derivative using a first-order difference quotient. For the derivative of $\boldsymbol{R}(t)$, there exist three kinds of the first-order difference quotient.

$$\begin{aligned}
\text{Forward quotient}: &\frac{d}{dt}\boldsymbol{R}(t) = \frac{\boldsymbol{R}(t+\Delta t) - \boldsymbol{R}(t)}{\Delta t} \\
\text{Central quotient}: &\frac{d}{dt}\boldsymbol{R}(t) = \frac{\boldsymbol{R}(t+\Delta t) - \boldsymbol{R}(t-\Delta t)}{2\Delta t} \\
\text{Backward quotient}: &\frac{d}{dt}\boldsymbol{R}(t) = \frac{\boldsymbol{R}(t) - \boldsymbol{R}(t-\Delta t)}{\Delta t}
\end{aligned} \tag{9}$$

where Δt is the time interval between two successive time instants. The above equation shows that the difference quotient is essentially a measure of the average rage of change of the rotation matrix $\boldsymbol{R}(t)$ over Δt. As the Δt approaches to zero, the limit of the difference quotient is thus the instantaneous derivative of $\boldsymbol{R}(t)$.

The forward Euler method is to update the rotation matrix in numerical integration using the forward quotient. Specifically, at the current time t, the rotation matrix at $t + \Delta t$ can be predicted by adding $\Delta t \cdot \boldsymbol{F}(\boldsymbol{R}, \boldsymbol{\omega}_R)$ to the rotation matrix $\boldsymbol{R}(t)$ by simply applying matrix addition, as shown below.

$$\boldsymbol{R}(t+\Delta t) = \boldsymbol{R}(t) + \Delta t \cdot \boldsymbol{F}(\boldsymbol{R}, \boldsymbol{\omega}_R) \tag{10}$$

where $\boldsymbol{R} = \boldsymbol{R}(t)$ and $\boldsymbol{\omega}_R = \boldsymbol{\omega}_R(t)$ are the values of the rotation matrix and the angular velocity taken at time t and keep constant during Δt. For a period from t_0 to t_f, the forward Euler method can be summarized as below.

Algorithm 1: Forward Euler Method

Input: Δt

Initialize the rotation matrix: $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}_0$

for $t \leftarrow t_0$ **to** $t_\mathrm{f} - \Delta t$ **do**

  Measure the angular velocity $\boldsymbol{\omega}_R \leftarrow \boldsymbol{\omega}_R(t)$

  Update the rotation matrix using Eq.(10): $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}(t) + \Delta t \cdot \boldsymbol{F}(\boldsymbol{R}, \boldsymbol{\omega}_R)$

  Update time: $t \leftarrow t + \Delta t$

end for

Return: sequence of numerical rotation matrices $\{\boldsymbol{R}(t)\}_{t_0}^{t_\mathrm{f}}$

3.2 Fourth-order Runge-Kutta Method

The Runge-Kutta method advances the numerical accuracy of the Euler method by approximating the first-order derivative with some improved formulas. For a ODE $\dot{y} = f(t, y)$, a general explicit Runge-Kutta method with s

steps (also known as 'orders') takes the following form [12].

$$y_{n+1} = y_n + h_n \sum_{r=1}^{s} b_r k_r \tag{11}$$

where h_n is the step-size at the iteration n, b_r is the coefficient, and k_r is the derivative in each stage defined as

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right) \\ k_r &= f\left(t_n + c_r h_n, y_n + h_n \sum_{j=1}^{r-1} a_{rj} k_j\right), \ 2 \leq r \leq s \end{aligned} \tag{12}$$

where these coefficients c_r, a_{rj}, and b_r can be displayed in a Butcher tableau of the following form.

$$\begin{array}{c|cccc} c_1 & 0 & & & \\ c_2 & a_{21} & 0 & & \\ \vdots & \vdots & \ddots & 0 & \\ c_s & a_{s1} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} \tag{13}$$

Comparison with the Euler method (10) shows that the Runge-Kutta method approximates the first-order derivative by a weighted summation of derivatives evaluated at different points in the time interval of the step-size. It is the improved approximation of the derivative that makes the Runge-Kutta method outperform the Euler method. Another observation from the Butcher tableau (13) is that the all non-zero coefficients a_{rj} are below the diagonal, meaning that the derivative k_r at r stage only depends on the derivatives at previous stages, and this is the feature of the explicit Runge-Kutta method [12]. When $s = 4$, the method is called 4th-order Runge-Kutta method which can be used to solve the kinematics (8) numerically as follows.

$$\boldsymbol{R}\left(t + \Delta t\right) = \boldsymbol{R}\left(t\right) + \frac{\Delta t}{6}\left(\boldsymbol{K}_1 + 2\boldsymbol{K}_2 + 2\boldsymbol{K}_3 + \boldsymbol{K}_4\right) \tag{14}$$

where these derivatives $\boldsymbol{K}_r, 1 \leq r \leq 4$ at different stages can be defined according to Eq.(12).

$$\begin{aligned} \boldsymbol{K}_1 &= \boldsymbol{F}\left(\boldsymbol{R}, \boldsymbol{\omega}_R\right) \\ \boldsymbol{K}_2 &= \boldsymbol{F}\left(\boldsymbol{R} + \Delta t \frac{\boldsymbol{K}_1}{2}, \boldsymbol{\omega}_R\right) \\ \boldsymbol{K}_3 &= \boldsymbol{F}\left(\boldsymbol{R} + \Delta t \frac{\boldsymbol{K}_2}{2}, \boldsymbol{\omega}_R\right) \\ \boldsymbol{K}_4 &= \boldsymbol{F}\left(\boldsymbol{R} + \Delta t \boldsymbol{K}_3, \boldsymbol{\omega}_R\right) \end{aligned} \tag{15}$$

where $\boldsymbol{R} = \boldsymbol{R}\left(t\right)$ and $\boldsymbol{\omega}_R = \boldsymbol{\omega}_R\left(t\right)$ are the values of the rotation matrix and the angular velocity taken at time t and keep constant during Δt. Because the function $\boldsymbol{F}$ does not depend on time t explicitly, the coefficients used in Eq.(14) and Eq.(15) are just parts of the coefficients of the standard 4th-order Runge-Kutta method whose Butcher tableau is shown below.

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array} \tag{16}$$

For a period from t_0 to t_f, the 4th-order Runge-Kutta method can be summarized as below.

Algorithm 2: Fourth-order Runge-Kutta Method

Input: Δt

Initialize the rotation matrix: $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}_0$

for $t \leftarrow t_0$ **to** $t_\mathrm{f} - \Delta t$ **do**

  Measure the angular velocity $\boldsymbol{\omega}_R \leftarrow \boldsymbol{\omega}_R(t)$

  Compute the derivatives $\boldsymbol{K}_r, 1 \leq r \leq 4$ at different stages using Eq.(15)

  Update the rotation matrix using Eq.(14): $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}(t) + \frac{\Delta t}{6}\left(\boldsymbol{K}_1 + 2\boldsymbol{K}_2 + 2\boldsymbol{K}_3 + \boldsymbol{K}_4\right)$

  Update time: $t \leftarrow t + \Delta t$

end for

Return: sequence of numerical rotation matrices $\{\boldsymbol{R}(t)\}_{t_0}^{t_\mathrm{f}}$

3.3 QR-factorization-based Method

The classical numerical methods, such as the forward Euler method and Runge-Kutta method, cannot guarantee that $\boldsymbol{R}(t)$ is still a rotation matrix after integration, since these methods are essentially translations not rotations. To preserve the matrix structure of rotation matrices, the author in [9] proposed to re-normalize $\boldsymbol{R}(t)$ obtained by the classical method. This is achieved by projecting $\boldsymbol{R}(t)$ onto $SO(3)$ using QR factorization, which decomposes the matrix $\boldsymbol{R}(t)$ into a product $\boldsymbol{R}(t) = \boldsymbol{Q}\bar{\boldsymbol{R}}$ of an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\bar{\boldsymbol{R}}$ [13].

There are three main methods to compute the QR factorization, namely the Gram–Schmidt process, the Householder reflections, and the use of the givens rotations. The first method is used in this project. Consider the matrix written in columns $\boldsymbol{R}(t) = [\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3]$ where $\boldsymbol{r}_i \in \mathbb{R}^{3\times 1},\ i = 1, 2, 3$. The orthogonal vectors forming the matrix $\boldsymbol{Q}$ can be computed by the following equations.

$$
\begin{aligned}
&\boldsymbol{u}_1 = \boldsymbol{r}_1,\ \boldsymbol{e}_1 = \frac{\boldsymbol{u}_1}{\|\boldsymbol{u}_1\|_2} \\
&\boldsymbol{u}_2 = \boldsymbol{r}_2 - \left(\boldsymbol{r}_2^T\boldsymbol{e}_1\right)\boldsymbol{e}_1,\ \boldsymbol{e}_2 = \frac{\boldsymbol{u}_2}{\|\boldsymbol{u}_2\|} \\
&\boldsymbol{u}_3 = \boldsymbol{r}_3 - \left(\boldsymbol{r}_3^T\boldsymbol{e}_1\right)\boldsymbol{e}_1 - \left(\boldsymbol{r}_3^T\boldsymbol{e}_2\right)\boldsymbol{e}_2,\ \boldsymbol{e}_3 = \frac{\boldsymbol{u}_3}{\|\boldsymbol{u}_3\|}
\end{aligned}
\tag{17}
$$

It is evident that these vectors $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, and $\boldsymbol{e}_3$ are orthogonal to each other, since $\boldsymbol{u}_i^T\boldsymbol{u}_j = \delta_{ij}$, for example, $\boldsymbol{u}_1^T \cdot \boldsymbol{u}_2 = \boldsymbol{r}_1^T\boldsymbol{r}_2 - \boldsymbol{r}_1^T\boldsymbol{e}_1\boldsymbol{e}_1^T\boldsymbol{r}_2 = 0$. The orthogonal matrix $\boldsymbol{Q}$ is therefore formulated as $\boldsymbol{Q} = [e_1, \boldsymbol{e}_2, \boldsymbol{e}_3]$, and the corresponding upper triangular matrix $\bar{\boldsymbol{R}}$ is formed as

$$
\bar{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{r}_1^T\boldsymbol{e}_1 & \boldsymbol{r}_2^T\boldsymbol{e}_1 & \boldsymbol{r}_3^T\boldsymbol{e}_1 \\ 0 & \boldsymbol{r}_2^T\boldsymbol{e}_2 & \boldsymbol{r}_3^T\boldsymbol{e}_2 \\ 0 & 0 & \boldsymbol{r}_3^T\boldsymbol{e}_3 \end{bmatrix}
\tag{18}
$$

The algorithm for a period from t_0 to t_f can be summarized as below.

Algorithm 3: QR-factorization-based Method

Input: Δt

Initialize the rotation matrix: $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}_0$

for $t \leftarrow t_0$ **to** $t_\mathrm{f} - \Delta t$ **do**

  Measure the angular velocity $\boldsymbol{\omega}_R \leftarrow \boldsymbol{\omega}_R(t)$

  Use the Algorithm 2 to solve for the numerical value of $\boldsymbol{R}(t)$

  Compute the QR factorization using Eq.(17) and Eq.(18) to obtain the matrices $\boldsymbol{Q}$ and $\bar{\boldsymbol{R}}$

  Compute the diagonal matrix using the signs of the diagonal elements of $\bar{\boldsymbol{R}}$: $\bar{\boldsymbol{R}}_d = \mathrm{diag}\left(\mathrm{sign}\left(\bar{\boldsymbol{R}}\right)\right)$

  Update the rotation matrix: $\boldsymbol{R}(t) \leftarrow \boldsymbol{Q}\bar{\boldsymbol{R}}_d$

  Update time: $t \leftarrow t + \Delta t$

end for

Return: sequence of numerical rotation matrices $\{\boldsymbol{R}(t)\}_{t_0}^{t_\mathrm{f}}$

3.4 Lie Group Integrator

It is acknowledged that the elements in a Lie group G evolves on the manifold $\mathcal{M}$ and the Lie algebra $\mathfrak{g}$ is defined as the tangent space at the identity T_eG of the manifold $\mathcal{M}$ [14]. This subsection first shows that any rotation matrix, i.e., $\boldsymbol{R} \in SO(3) \doteq \left\{\boldsymbol{A} \in \mathbb{R}^{3\times 3} \,\middle|\, \boldsymbol{A}\boldsymbol{A}^T = \mathbb{I}_{3\times 3},\ \det(\boldsymbol{A}) = 1\right\}$, evolve on an unit sphere $S^2 \subset \mathbb{R}^3$, and then details the design procedures of the Lie group integrator that preserves the matrix structures of rotation matrices.

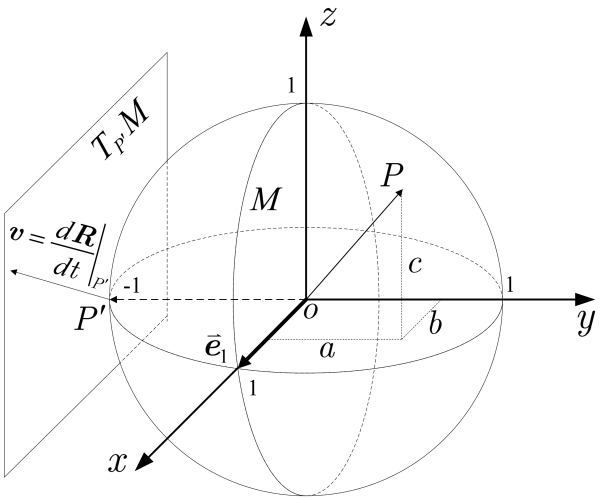


Figure 1: Illustration of an unit sphere and its tangent space.

Consider a three-dimensional vector $\boldsymbol{p} = [a, b, c]^T \in \mathbb{R}^3$ subject to the unit norm $a^2 + b^2 + c^2 = 1$. Due to the constraint, the vector $\boldsymbol{p}$ should be confined to an unit sphere as shown in Fig.1. There exists a rotation matrix $\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3] \in \mathbb{R}^{3\times 3}$ that is able to rotate the basis vector $\bar{\boldsymbol{e}}_1 = [1, 0, 0]^T$ to the vector $\boldsymbol{p}$ [15]. To construct such rotation matrix, the corresponding column vectors $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$ can be formulated as follows.

$$\begin{aligned} \boldsymbol{r}_1 &= \boldsymbol{p} \\ \boldsymbol{r}_2 &= \frac{\boldsymbol{r}_1 \times \bar{\boldsymbol{e}}_1}{\|\boldsymbol{r}_1 \times \bar{\boldsymbol{e}}_1\|} \\ \boldsymbol{r}_3 &= \frac{\boldsymbol{r}_1 \times \boldsymbol{r}_2}{\|\boldsymbol{r}_1 \times \boldsymbol{r}_2\|} \end{aligned} \tag{19}$$

The rotation matrix is finally formed as $\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_3, \boldsymbol{r}_2]$ by permuting $\boldsymbol{r}_2$ and $\boldsymbol{r}_3$ to guarantee $\det(\boldsymbol{R}) = 1$, which takes the following form [15].

$$\boldsymbol{R} = \begin{bmatrix} a & \sqrt{b^2+c^2} & 0 \\ b & \frac{-ab}{\sqrt{b^2+c^2}} & \frac{c}{\sqrt{b^2+c^2}} \\ c & \frac{-ac}{\sqrt{b^2+c^2}} & \frac{-b}{\sqrt{b^2+c^2}} \end{bmatrix} \tag{20}$$

The above matrix satisfies the two matrix structures, i.e., $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_3$ and $\det(\boldsymbol{R}) = 1$, and makes the rotation $\boldsymbol{p} = \boldsymbol{R}\bar{\boldsymbol{e}}_1$ hold for all a, b, and c that satisfy the constraint $a^2 + b^2 + c^2 = 1$. This proves that the rotation matrix evolves on the manifold of $S^2 \subset \mathbb{R}^3$ and is confined to the Lie group $SO(3)$.

The next is to show the geometric relation between Lie group and Lie algebra and how this relationship is used to derive the Lie group integrator. Without loss of generality, consider the rotation matrix $\boldsymbol{R}(t + \Delta t)$ at time $t + \Delta t$ and expand it at $\boldsymbol{R}(t)$ using a Taylor series.

$$\boldsymbol{R}(t+\Delta t) = \boldsymbol{R}(t) + \Delta t \frac{d}{dt}\boldsymbol{R}(t) + \frac{\Delta t^2}{2!}\frac{d^2}{dt^2}\boldsymbol{R}(t) + \cdots + \frac{\Delta t^n}{n!}\frac{d^n}{dt^n}\boldsymbol{R}(t) + \cdots \tag{21}$$

Define the vector field as

$$\boldsymbol{v} = \frac{d}{dt} \tag{22}$$

which is exactly the tangent space of the sphere as shown in Fig.1, such that

$$\frac{d^n}{dt^n}\boldsymbol{R}(t) = \boldsymbol{v}^n \boldsymbol{R}(t). \tag{23}$$

Using the above definitions and the matrix exponential,

$$\mathbf{exp}(\boldsymbol{A}) = \mathbb{I}_3 + \boldsymbol{A} + \frac{\boldsymbol{A}^2}{2!} + \cdots + \frac{\boldsymbol{A}^n}{n!} + \cdots \tag{24}$$

we can rewrite Eq.(21) as below.

$$\begin{aligned} \boldsymbol{R}(t+\Delta t) &= \left(\mathbb{I}_3 + \Delta t\boldsymbol{v} + \frac{\Delta t^2}{2!}\boldsymbol{v}^2 + \cdots \frac{\Delta t^n}{n!}\boldsymbol{v}^n + \cdots\right)\boldsymbol{R}(t) \\ &= \mathbf{exp}(\Delta t\boldsymbol{v})\,\boldsymbol{R}(t) \end{aligned} \tag{25}$$

Assume that the angular velocity keeps constant during a small time interval Δt such that $\boldsymbol{v}^n\boldsymbol{R} = \boldsymbol{v}^{n-1}(\boldsymbol{R}\boldsymbol{\omega}_R^\wedge) = \left(\boldsymbol{v}^{n-1}\boldsymbol{R}\right)\boldsymbol{\omega}_R^\wedge = \boldsymbol{R}(\boldsymbol{\omega}_R^\wedge)^{n-1}\boldsymbol{\omega}_R^\wedge = \boldsymbol{R}(\boldsymbol{\omega}_R^\wedge)^n$ which is achieved by considering the kinematics (7). Plugging this relation into Eq.(25) yields

$$\begin{aligned} \boldsymbol{R}(t+\Delta t) &= \boldsymbol{R}(t)\left(I_3 + \Delta t\boldsymbol{\omega}_R^\wedge + \frac{\Delta t^2}{2!}(\boldsymbol{\omega}_R^\wedge)^2 + \cdots \frac{\Delta t^n}{n!}(\boldsymbol{\omega}_R^\wedge)^n + \cdots\right) \\ &= \boldsymbol{R}(t)\,\mathbf{exp}(\Delta t\boldsymbol{\omega}_R^\wedge) \end{aligned} \tag{26}$$

where $\boldsymbol{\omega}_R^\wedge$ is the tangent space of the rotation matrix $\boldsymbol{R}$ and referred to as the Lie algebra $\mathfrak{so}(3)$ of the Lie group $SO(3)$. The matrix exponential $\mathbf{exp}(\Delta t\boldsymbol{\omega}_R^\wedge)$ is also a rotation matrix, since the following matrix structures hold.

$$\mathbf{exp}(\Delta t\boldsymbol{\omega}_R^\wedge)\cdot\mathbf{exp}\left(\Delta t(\boldsymbol{\omega}_R^\wedge)^T\right) \xlongequal{\boldsymbol{\omega}_R^\wedge(\boldsymbol{\omega}_R^\wedge)^T = (\boldsymbol{\omega}_R^\wedge)^T\boldsymbol{\omega}_R^\wedge} \mathbf{exp}\left(\Delta t\left(\boldsymbol{\omega}_R^\wedge - \boldsymbol{\omega}_R^\wedge\right)\right) = \mathbb{I}_{3\times 3} \tag{27a}$$

$$\det\left(\mathbf{exp}(\Delta t\boldsymbol{\omega}_R^\wedge)\right) = \prod_{i=1}^{3}\lambda_i\left(\mathbf{exp}(\Delta t\boldsymbol{\omega}_R^\wedge)\right) = \mathbf{exp}\left(\Delta t\sum_{i=1}^{3}\lambda_i\left(\boldsymbol{\omega}_R^\wedge\right)\right) = \mathbf{exp}\left(\Delta t\cdot\mathrm{trace}\left(\boldsymbol{\omega}_R^\wedge\right)\right) = 1 \tag{27b}$$

It turns out that the matrix exponential of a Lie algebra element is the Lie group element, in turn, the logarithm of a Lie group element is the corresponding Lie algebra element. This correspondence of Lie algebra and Lie group is illustrated in Fig.2 where the matrix exponential and the logarithm are referred to as the exponential map and the logarithm map, respectively.

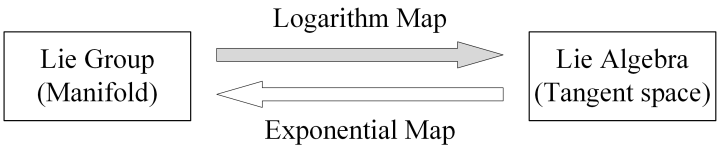


Figure 2: Illustration of the correspondence of Lie group and Lie algebra.

Equation (24) says that in general the amount of terms in the matrix exponential is infinite, which makes it difficult to calculate the matrix exponential accurately in computers. Fortunately, the following matrix properties of skew-symmetric matrices can be used to make the amount of terms finite.

$$\left\|\Delta t\cdot\boldsymbol{\omega}_R^\wedge\right\|_2^2 = \Delta t^2\left\|\boldsymbol{\omega}_R\right\|_2^2 \tag{28a}$$

$$\left(\boldsymbol{\omega}_R^\wedge\right)^{2n} = (-1)^{n-1}\left\|\boldsymbol{\omega}_R\right\|_2^{2n-2}\left(\boldsymbol{\omega}_R^\wedge\right)^2,\ n\in\mathbb{Z}^+ \tag{28b}$$

$$\left(\boldsymbol{\omega}_R^{\wedge}\right)^{2n+1} = (-1)^n \left\|\boldsymbol{\omega}_R\right\|_2^{2n} \boldsymbol{\omega}_R^{\wedge},\ n \in \mathbb{Z}^+ \tag{28c}$$

Therefore, we can rewrite $\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_R^{\wedge}\right)$ as

$$\begin{aligned}
\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_R^{\wedge}\right) = \mathbb{I}_{3\times 3} &+ \Delta t \boldsymbol{\omega}_R^{\wedge} + \frac{\Delta t^2}{2!}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 - \frac{\Delta t^3 \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^2}{3!}\boldsymbol{\omega}_R^{\wedge} - \frac{\Delta t^4 \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^2}{4!}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 \\
&+ \frac{\Delta t^5 \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^4}{5!}\boldsymbol{\omega}_R^{\wedge} + \frac{\Delta t^6 \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^4}{6!}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 + \cdots \\
&+ (-1)^{n-1}\frac{\Delta t^{2n} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n-2}}{(2n)!}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 \\
&+ (-1)^{n}\frac{\Delta t^{2n+1} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n}}{(2n+1)!}\boldsymbol{\omega}_R^{\wedge} + \cdots,\ n \in \mathbb{Z}^+.
\end{aligned} \tag{29}$$

Given the Taylor series

$$\begin{aligned}
\sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \\
\cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots
\end{aligned} \tag{30}$$

Equation (29) can be further simplified as

$$\begin{aligned}
\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_R^{\wedge}\right) &= \mathbb{I}_{3\times 3} + \sum_{n=1}^{\infty}(-1)^{n-1}\frac{\Delta t^{2n} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n-2}}{(2n)!}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 + \sum_{n=0}^{\infty}(-1)^{n}\frac{\Delta t^{2n+1} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n}}{(2n+1)!}\boldsymbol{\omega}_R^{\wedge} \\
&= \mathbb{I}_{3\times 3} + \sum_{n=1}^{\infty}(-1)^{n-1}\frac{\Delta t^{2n} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n}}{(2n)!\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^2}\left(\boldsymbol{\omega}_R^{\wedge}\right)^2 + \sum_{n=0}^{\infty}(-1)^{n}\frac{\Delta t^{2n+1} \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^{2n+1}}{(2n+1)!\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2}\boldsymbol{\omega}_R^{\wedge} \\
&= \boxed{\mathbb{I}_{3\times 3} + \left[1 - \cos\left(\Delta t \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2\right)\right]\frac{\left(\boldsymbol{\omega}_R^{\wedge}\right)^2}{\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^2} + \sin\left(\Delta t \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2\right)\frac{\boldsymbol{\omega}_R^{\wedge}}{\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2}}
\end{aligned} \tag{31}$$

The assumption of the constant angular velocity may not hold in practice, especially for robots performing agile maneuvers. To tackle this issue, the angular velocity $\boldsymbol{\omega}_R(t)$ at time t is replaced by the average angular velocity defined as

$$\bar{\boldsymbol{\omega}}_R(t) = \frac{1}{2}\left(\boldsymbol{\omega}_R(t) + \boldsymbol{\omega}_R(t + \Delta t)\right) \tag{32}$$

where $\boldsymbol{\omega}_R(t + \Delta t)$ can be obtained by the model prediction for online numerical integration, or by the available data-set for offline numerical integration.

Based on Eq.(26), the Lie group integrator algorithm for a period from t_0 to t_f can be summarized as below.

Algorithm 4: Lie Group Integrator Method

Input: Δt

Initialize the rotation matrix: $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}_0$

for $t \leftarrow t_0$ **to** $t_\mathrm{f} - \Delta t$ **do**

  Measure the angular velocity $\boldsymbol{\omega}_R \leftarrow \boldsymbol{\omega}_R(t)$

  Compute the average angular velocity using Eq.(32): $\bar{\boldsymbol{\omega}}_R(t) \leftarrow \frac{1}{2}\left(\boldsymbol{\omega}_R(t) + \boldsymbol{\omega}_R(t + \Delta t)\right)$

  Compute the matrix exponential of $\Delta t \cdot \bar{\boldsymbol{\omega}}_R^{\wedge}$ using Eq.(31):

$$\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_R^{\wedge}\right) = \mathbb{I}_{3\times 3} + \left[1 - \cos\left(\Delta t \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2\right)\right]\frac{\left(\boldsymbol{\omega}_R^{\wedge}\right)^2}{\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2^2} + \sin\left(\Delta t \left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2\right)\frac{\boldsymbol{\omega}_R^{\wedge}}{\left\|\boldsymbol{\omega}_R^{\wedge}\right\|_2}$$

  Update the rotation matrix using Eq.(26): $\boldsymbol{R}(t) \leftarrow \boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \bar{\boldsymbol{\omega}}_R^{\wedge}\right)$

  Update time: $t \leftarrow t + \Delta t$

end for

Return: sequence of numerical rotation matrices $\left\{\boldsymbol{R}(t)\right\}_{t_0}^{t_\mathrm{f}}$

4 Theoretical Comparisons

This section analyzes why the classical methods fail to preserve the matrix structures of rotation matrices and how they can be improved to have the comparable accuracy to the Lie group integrator, from the perspective of Taylor series. Then, a planar rotation example is used to geometrically demonstrate the effectiveness of the Lie group integrator and the improvement on accuracy of the Runge-Kutta method compare with the Euler method.

4.1 Limitations of Classical Methods and Why Lie Group Integrator

Let us first examine the forward Euler method in terms of the ability to preserve the matrix structure $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_{3\times 3}$. According to Eq.(10), the product of the updated matrix $\boldsymbol{R}(t+\Delta t)$ and its transpose is given by

$$\begin{aligned}\boldsymbol{R}(t+\Delta t)\,\boldsymbol{R}^T(t+\Delta t) &= \left(\boldsymbol{R}(t)+\Delta t\boldsymbol{F}(\boldsymbol{R},\boldsymbol{\omega}_R)\right)\left(\boldsymbol{R}(t)+\Delta t\boldsymbol{F}(\boldsymbol{R},\boldsymbol{\omega}_R)\right)^T \\ &= \boldsymbol{R}(t)\,\boldsymbol{R}^T(t)+\Delta t\boldsymbol{R}(t)\,\boldsymbol{F}^T(\boldsymbol{R},\boldsymbol{\omega}_R)+\Delta t\boldsymbol{F}(\boldsymbol{R},\boldsymbol{\omega}_R)\,\boldsymbol{R}^T(t) \\ &\quad +\Delta t^2\boldsymbol{F}(\boldsymbol{R},\boldsymbol{\omega}_R)\,\boldsymbol{F}^T(\boldsymbol{R},\boldsymbol{\omega}_R)\end{aligned} \tag{33}$$

Plugging $\boldsymbol{F}(\boldsymbol{R},\boldsymbol{\omega}_R) = \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge(t)$ into Eq.(33) and considering $(\boldsymbol{\omega}_R^\wedge)^T = -\boldsymbol{\omega}_R^\wedge$ yield

$$\boldsymbol{R}(t+\Delta t)\,\boldsymbol{R}^T(t+\Delta t) = \boldsymbol{R}(t)\,\boldsymbol{R}^T(t)-\Delta t^2\boldsymbol{R}(t)\,(\boldsymbol{\omega}_R^\wedge)^2\boldsymbol{R}^T(t) \tag{34}$$

Equation (34) says that $\boldsymbol{R}(t+\Delta t)\,\boldsymbol{R}^T(t+\Delta t) = \mathbb{I}_{3\times 3}$ doe not hold in general. Even if it may hold when $\boldsymbol{R}(t)\,\boldsymbol{R}^T(t) = \mathbb{I}_{3\times 3}+\Delta t^2\boldsymbol{R}(t)\,(\boldsymbol{\omega}_R^\wedge)^2\boldsymbol{R}^T(t)$, however, the equation $\boldsymbol{R}(t+2\Delta t)\,\boldsymbol{R}^T(t+2\Delta t) = \mathbb{I}_{3\times 3}$ still does not hold at next iteration. This is due to the translation nature of the forward Euler method, such that the updated matrix $\boldsymbol{R}(t+\Delta t)$ no longer evolves on the manifold of the Lie group $SO(3)$.

Compared with the forward Euler method, the 4th-order Runge-Kutta method can improve the numerical accuracy. According to Eq.(15), the derivatives at different stages can be obtained as follows.

$$\begin{aligned}\boldsymbol{K}_1 &= \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge \\ \boldsymbol{K}_2 &= \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge+\frac{\boldsymbol{R}(t)}{2}\Delta t(\boldsymbol{\omega}_R^\wedge)^2 \\ \boldsymbol{K}_3 &= \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge+\frac{\boldsymbol{R}(t)}{2}\Delta t(\boldsymbol{\omega}_R^\wedge)^2+\frac{\boldsymbol{R}(t)}{4}\Delta t^2(\boldsymbol{\omega}_R^\wedge)^3 \\ \boldsymbol{K}_4 &= \boldsymbol{R}(t)\,\boldsymbol{\omega}_R^\wedge+\boldsymbol{R}(t)\,\Delta t(\boldsymbol{\omega}_R^\wedge)^2+\frac{\boldsymbol{R}(t)}{2}\Delta t^2(\boldsymbol{\omega}_R^\wedge)^3+\frac{\boldsymbol{R}(t)}{4}\Delta t^3(\boldsymbol{\omega}_R^\wedge)^4\end{aligned} \tag{35}$$

Plugging Eq.(35) into Eq.(14) yields

$$\boldsymbol{R}(t+\Delta t) = \boldsymbol{R}(t)\left(\mathbb{I}_{3\times 3}+\Delta t\boldsymbol{\omega}_R^\wedge+\frac{\Delta t^2(\boldsymbol{\omega}_R^\wedge)^2}{2!}+\frac{\Delta t^3(\boldsymbol{\omega}_R^\wedge)^3}{3!}+\frac{\Delta t^4(\boldsymbol{\omega}_R^\wedge)^4}{4!}\right). \tag{36}$$

Equation (36) can be generalized to the kth-order Runge-Kutta method that updates the matrix $\boldsymbol{R}(t+\Delta t)$ as

$$\boldsymbol{R}(t+\Delta t) = \boldsymbol{R}(t)\left(\mathbb{I}_{3\times 3}+\sum_{n=1}^{k}\frac{1}{n!}(\Delta t\boldsymbol{\omega}_R^\wedge)^n\right) \tag{37}$$

Compared with the Lie group integrator expanded by a Taylor series

$$\begin{aligned}\boldsymbol{R}(t+\Delta t) &= \boldsymbol{R}(t)\,\mathbf{exp}\,(\Delta t\boldsymbol{\omega}_R^\wedge) \\ &= \boldsymbol{R}(t)\left(\mathbb{I}_{3\times 3}+\sum_{n=1}^{\infty}\frac{1}{n!}(\Delta t\boldsymbol{\omega}_R^\wedge)^n\right),\end{aligned} \tag{38}$$

the kth-order Runge-Kutta method (37) essentially corresponds to the first k terms of the Lie group integrator with

the following truncation error $\boldsymbol{E}_k$.

$$\boldsymbol{E}_k = \boldsymbol{R}(t) \sum_{n=k+1}^{\infty} \frac{1}{n!} (\Delta t \boldsymbol{\omega}_R^{\wedge})^n \tag{39}$$

The Lie group integrator is able to preserve these two matrix structures perfectly. Without loss of generality, consider two successive iterations with two different angular velocities $\boldsymbol{\omega}_{R1}$ and $\boldsymbol{\omega}_{R2}$ respectively and assume that the initial matrix $\boldsymbol{R}(t)$ is a rotation matrix. The calculations are shown below.

$$\begin{aligned}
\boldsymbol{R}(t+2\Delta t)\,\boldsymbol{R}^T(t+2\Delta t) &= \boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R2}^{\wedge}\right)\mathbf{exp}\left(\Delta t (\boldsymbol{\omega}_{R2}^{\wedge})^T\right)\mathbf{exp}\left(\Delta t (\boldsymbol{\omega}_{R1}^{\wedge})^T\right)\boldsymbol{R}^T(t) \\
&= \boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R2}^{\wedge}\right)\mathbf{exp}\left(-\Delta t \boldsymbol{\omega}_{R2}^{\wedge}\right)\mathbf{exp}\left(-\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\boldsymbol{R}^T(t) \\
&= \boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\mathbb{I}_{3\times 3}\mathbf{exp}\left(-\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\boldsymbol{R}^T(t) \\
&= \boldsymbol{R}(t)\,\boldsymbol{R}^T(t) \\
&= \mathbb{I}_{3\times 3}
\end{aligned} \tag{40a}$$

$$\begin{aligned}
\det\left(\boldsymbol{R}(t+2\Delta t)\right) &= \det\left(\boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R2}^{\wedge}\right)\right) \\
&= \det\left(\boldsymbol{R}(t)\right)\det\left(\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R1}^{\wedge}\right)\right)\det\left(\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_{R2}^{\wedge}\right)\right) \\
&= \mathbf{exp}\left(\Delta t \cdot \mathrm{trace}\left(\boldsymbol{\omega}_{R1}^{\wedge}\right)\right)\mathbf{exp}\left(\Delta t \cdot \mathrm{trace}\left(\boldsymbol{\omega}_{R2}^{\wedge}\right)\right) \\
&= 1
\end{aligned} \tag{40b}$$

Equations (40a) and (40b) can also be generalized to n iterations of the Lie group integrator, provided that the initial matrix $\boldsymbol{R}(t)$ is a rotation matrix. This is because the exponential map of the Lie algebra $\boldsymbol{\omega}_R^{\wedge}$ is the Lie group $SO(3)$ and the product of two rotation matrices $\boldsymbol{R}(t)\,\mathbf{exp}\left(\Delta t \boldsymbol{\omega}_R^{\wedge}\right)$ is still a rotation matrix. Therefore, *the truncation error between the classical methods and the Lie group integrator measures the numerical accuracy*, **smaller the error is, higher the accuracy will be**. This observation validates that the 4th-order Runge-Kutta method is more accurate than the forward Euler method, since the latter is essentially the 1st-order Runge-Kutta method with a larger truncation error $\boldsymbol{E}_1 > \boldsymbol{E}_4$. This further implies that the numerical accuracy of the Runge-Kutta method can be improved by increasing its order, since the higher order enables it to approximate the Lie group integrator better. Theoretically, if the order approaches to infinity, we have

$$\lim_{k\to\infty} \boldsymbol{E}_k = \lim_{k\to\infty} \left[\boldsymbol{R}(t) \sum_{n=k+1}^{\infty} \frac{1}{n!} (\Delta t \boldsymbol{\omega}_R^{\wedge})^n\right] = \mathbf{0}. \tag{41}$$

In that case, the infinite-order Runge-Kutta method evolves to the Lie group integrator, which however is difficult to implement in computers.

4.2 Geometric Comparison of Lie Group Integrator and Classical Methods

In this subsection, a planar rotation example is used to illustrate the structure-preserving feature of the Lie group integrator, and how a 2nd-order Runge-Kutta method can better approximate the Lie group integrator than the forward Euler method.

Consider a planar rotation matrix parameterized by the angle θ

$$\boldsymbol{R}(t) = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) \\ \sin\theta(t) & \cos\theta(t) \end{bmatrix}. \tag{42}$$

This rotation matrix evolves on a unit circle $S^1 = \left\{(x,y) \in \mathbb{R}^2 \,\middle|\, x^2 + y^2 = 1\right\} \subset \mathbb{R}^2$ and its kinematics is given by

$$\frac{d}{dt}\boldsymbol{R}(t) = \omega^{\wedge}\boldsymbol{R}(t) \tag{43}$$

where the skew-symmetric matrix $\omega^{\wedge} \in \mathbb{R}^{2\times 2}$ is defined as

$$\omega^{\wedge} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \dot{\theta} \tag{44}$$

To simplify the analysis, assume that the angular velocity satisfies $\dot{\theta} = 1$ and the initial angle is zero, $\theta_0 = 0$. In that case, the kinematics (43) at the initial time simplifies to

$$\dot{\boldsymbol{R}}(0) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} = [\boldsymbol{k}_1^{e_1}, \boldsymbol{k}_1^{e_2}] \tag{45}$$

which can be visualized by two vectors $\boldsymbol{k}_1^{e_1}$ and $\boldsymbol{k}_1^{e_2}$, as shown in Fig.3.

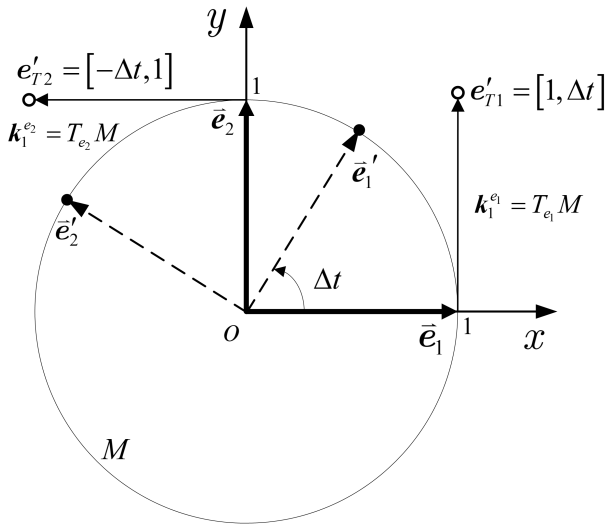


Figure 3: Illustration of the numerical accuracy of the Lie group integrator and the forward Euler method. The vectors $T_{\boldsymbol{e}_1} M$ and $T_{\boldsymbol{e}_2} M$ are the tangent spaces of the circle M at $\boldsymbol{e}_1 = [1, 0]^T$ and $\boldsymbol{e}_1 = [0, 1]^T$, respectively. The points $\boldsymbol{e}_1 = [1, 0]^T$ and $\boldsymbol{e}_1 = [0, 1]^T$ construct the initial rotation matrix $\boldsymbol{R}(0) = [\boldsymbol{e}_1, \boldsymbol{e}_2] = \mathbb{I}_{2\times 2}$ while the points $\boldsymbol{e}_1{}' = [\cos\Delta t, \sin\Delta t]^T$ and $\boldsymbol{e}_2{}' = [-\sin\Delta t, \cos\Delta t]^T$ construct the rotation matrix $\boldsymbol{R}(\Delta t) = [\boldsymbol{e}_1{}', \boldsymbol{e}_2{}']$ updated by the Lie group integrator. The matrix $\boldsymbol{A} = [\boldsymbol{e}'_{T1}, \boldsymbol{e}'_{T2}]$ is obtained by the forward Euler method.

After a period of time Δt, the Lie group integrator updates the rotation matrix by

$$\boldsymbol{R}(\Delta t) = \mathbf{exp}(\Delta t \cdot \omega^{\wedge}) \tag{46}$$

where the matrix exponential can be computed using Eq.(26)

$$\mathbf{exp}(\Delta t \cdot \omega^{\wedge}) = \mathbb{I}_{2\times 2} + \sin\Delta t \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} + (1 - \cos\Delta t) \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}^2. \tag{47}$$

Plugging Eq.(47) into Eq.(46) yields

$$\boldsymbol{R}(\Delta t) = \begin{bmatrix} \cos\Delta t & -\sin\Delta t \\ \sin\Delta t & \cos\Delta t \end{bmatrix} \tag{48}$$

which is a rotation matrix, since

$$\boldsymbol{R}(\Delta t)\, \boldsymbol{R}^T(\Delta t) = \begin{bmatrix} \sin^2\Delta t + \cos^2\Delta t & 0 \\ 0 & \sin^2\Delta t + \cos^2\Delta t \end{bmatrix} = \mathbb{I}_{2\times 2} \tag{49a}$$

and

$$\det(\boldsymbol{R}(\Delta t)) = \cos^2\Delta t + \sin^2\Delta t = 1. \tag{49b}$$

The matrix $\boldsymbol{R}(\Delta t)$ can be represented by two points $\boldsymbol{e}_1' = [\cos\Delta t, \sin\Delta t]^T$ and $\boldsymbol{e}_2' = [-\sin\Delta t, \cos\Delta t]^T$ which evolve on the unit circle, as illustrated in Fig.3.

As a comparison, the forward Euler method updates the rotation matrix by

$$\begin{aligned}\boldsymbol{A} &= \boldsymbol{R}(0) + \Delta t \cdot \omega^{\wedge}\boldsymbol{R}(0) \\ &= \begin{bmatrix} 1 & -\Delta t \\ \Delta t & 1 \end{bmatrix}\end{aligned} \tag{50}$$

Figure.3 shows that the resultant matrix $\boldsymbol{A}$ is represented by two points $\boldsymbol{e}_{T1}' = [1, \Delta t]^T$ and $\boldsymbol{e}_{T2}' = [-\Delta t, 1]^T$ which are translated from the initial points $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ by the distance Δt along their respective tangent spaces. These two points $\boldsymbol{e}_{T1}'$ and $\boldsymbol{e}_{T2}'$ illustrate the translation nature of the forward Euler method such that the matrix $\boldsymbol{A}$ is no longer a rotation matrix. Furthermore, as time increases, the points $\boldsymbol{e}_{T1}'$ and $\boldsymbol{e}_{T2}'$ deviate even farther away from the unit circle, making it less likely for the matrix $\boldsymbol{A}$ to be a rotation matrix.

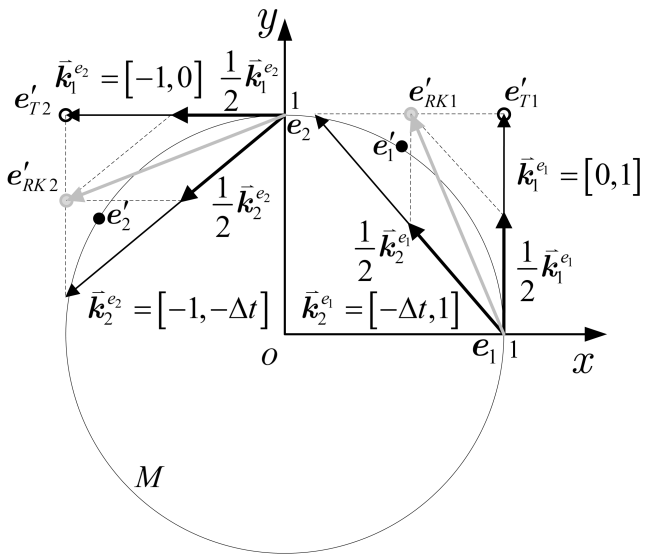


Figure 4: Illustration of the improvement of the 2nd-order Runge-Kutta method over the forward Euler method. The two gray points construct the matrix $\boldsymbol{B} = [\boldsymbol{e}'_{RK1}, \boldsymbol{e}'_{RK2}]$ obtained by the 2nd-order Runge-Kutta method.

As discussed in the previous subsection, the higher-order Runge-Kutta method is able to improve the numerical accuracy over the forward Euler method by decreasing the truncation error. In what follows, this improvement is explained from the geometric perspective.

Based on Eq.(12), the 2nd-order Runge-Kutta method updates the rotation matrix by

$$\boldsymbol{B} = \boldsymbol{R}(0) + \frac{\Delta t}{2}(\boldsymbol{K}_1 + \boldsymbol{K}_2) \tag{51}$$

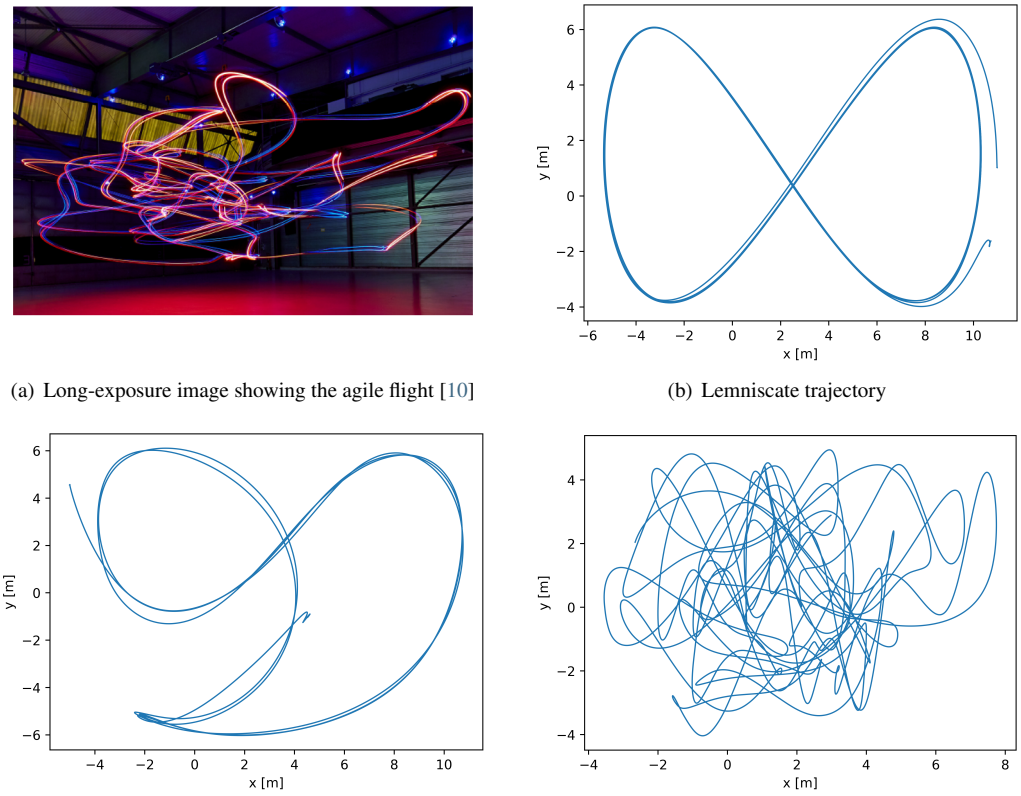
where $\boldsymbol{K}_1 = \omega^{\wedge}\boldsymbol{R}(0)$ and $\boldsymbol{K}_2$ is defined by

$$\begin{aligned}\boldsymbol{K}_2 &= \omega^{\wedge}(\boldsymbol{R}(0) + \Delta t \cdot K_1) \\ &= \begin{bmatrix} -\Delta t & -1 \\ 1 & -\Delta t \end{bmatrix}\end{aligned} \tag{52}$$

The stage derivative matrix $\boldsymbol{K}_2$ can be illustrated by two vectors $\boldsymbol{k}_2^{e1} = [-\Delta t, 1]^T$ and $\boldsymbol{k}_2^{e2} = [-1, -\Delta t]^T$. Figure.4 shows that $\boldsymbol{K}_2$ **is a correction term which makes the resultant matrix $\boldsymbol{B}$ move towards the unit circle**. This correction geometrically explains why the higher-order Runge-Kutta method is advantageous over the forward Euler method in terms of the structure-preserving ability.

5 Simulation

The numerical comparisons between the four algorithms were done using a real flight data-set [10]. Such a data-set was collected by making a quadrotor unmanned aerial vehicle (UAV) track several agile trajectories such as lemniscate, race, and random trajectories, as shown in Fig.5. The attitude and angular velocities of the quadrotor UAV vary significantly in these trajectories, making it challenging for the classical methods to obtain accurate numerical solutions. On the other hand, these challenging trajectories are good examples of highlighting the advantages of the Lie group integrator over the classical methods.



(a) Long-exposure image showing the agile flight [10]

(b) Lemniscate trajectory

(c) Race trajectory

(d) Random trajectory

Figure 5: Agile trajectories generated by the data-set [10] and used in the numerical comparisons

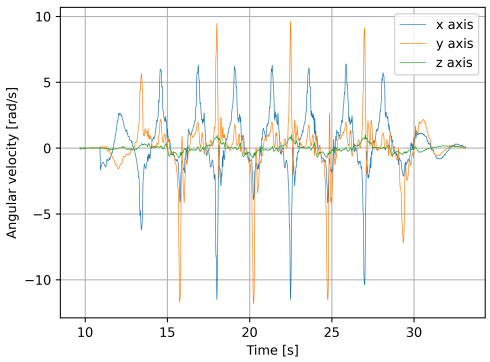
The numerical accuracy in simulation is defined as the ability of these algorithms to preserve the two matrix structures,i.e., $\det(\boldsymbol{R}) = 1$ and $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_{3\times 3}$. The latter is measured by the following attitude error function Ψ proposed in [16].

$$\begin{aligned} \Psi(\boldsymbol{R}) &= \frac{1}{2}\text{trace}\left(\mathbb{I}_{3\times 3} - \boldsymbol{R}^T\boldsymbol{R}\right) \\ \Psi(\boldsymbol{R}, \boldsymbol{R}_{\text{gt}}) &= \frac{1}{2}\text{trace}\left(\mathbb{I}_{3\times 3} - \boldsymbol{R}_{\text{gt}}^T\boldsymbol{R}\right) \end{aligned} \tag{53}$$

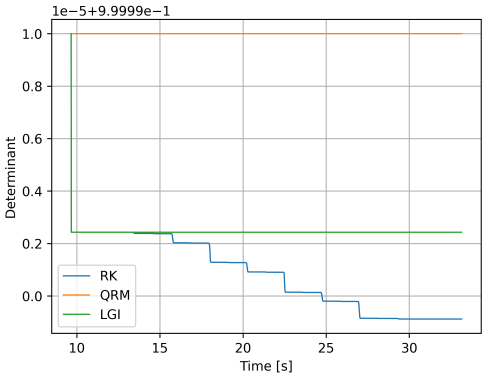
where $\Psi(\boldsymbol{R})$ is the self-attitude error function and $\Psi(\boldsymbol{R}, \boldsymbol{R}_{\text{gt}})$ is the function that measures the attitude error between the numerical solution $\boldsymbol{R}$ and the ground truth rotation matrix $\boldsymbol{R}_{\text{gt}}$ defined by quaternion $\boldsymbol{q} \doteq [q_0, q_1, q_2, q_3]$.

$$\boldsymbol{R}_{\text{gt}} = \begin{bmatrix} 2\left(q_0^2 + q_1^2\right) - 1 & 2q_1q_2 - 2q_0q_3 & 2q_0q_2 + 2q_1q_3 \\ 2q_0q_3 + 2q_1q_2 & 2\left(q_0^2 + q_2^2\right) - 1 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_0q_2 & 2q_0q_1 + 2q_2q_3 & 2\left(q_0^2 + q_3^2\right) - 1 \end{bmatrix} \tag{54}$$

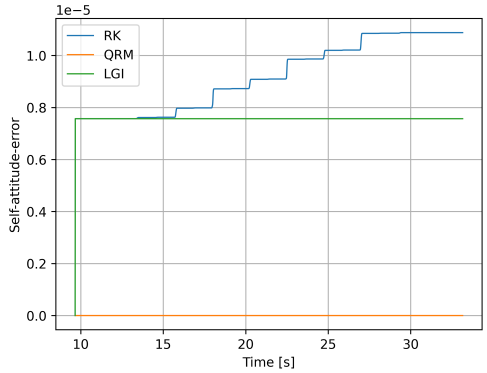
The step-size was set as 0.01s, the simulation code was programmed in `Python`, and the main code is attached in Appendix 8. The simulation results using the three kinds of trajectories are shown below. The labels 'RK', 'QRM', and 'LGI' denote '4th-order Runge-Kutta method', 'QR-factorization-based method', and 'Lie group integrator method', respectively



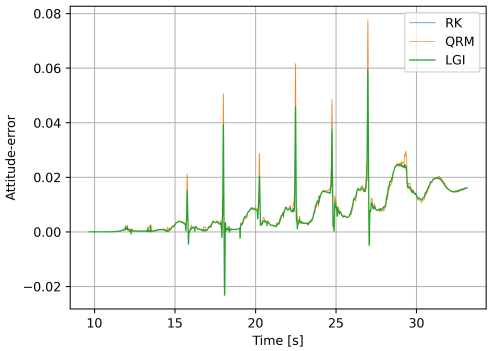
(a) Angular velocities of the lemniscate trajectory



(b) Determinant of the numerical solution $\boldsymbol{R}$

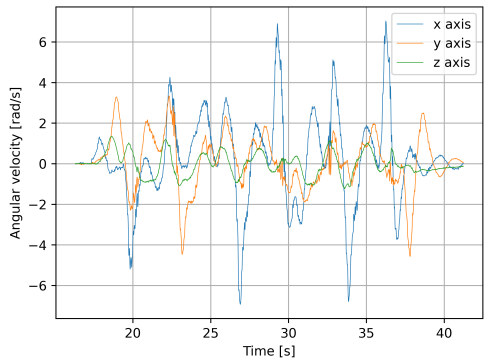


(c) Self-attitude error function $\Psi(\boldsymbol{R})$

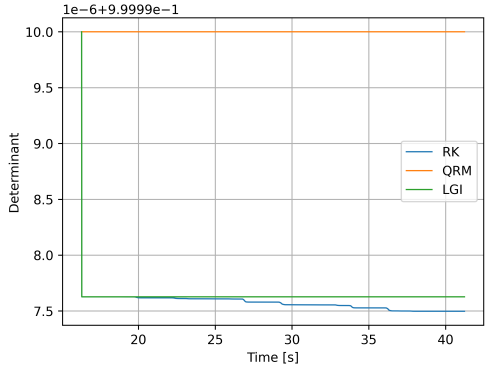


(d) Attitude error $\Psi(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}})$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

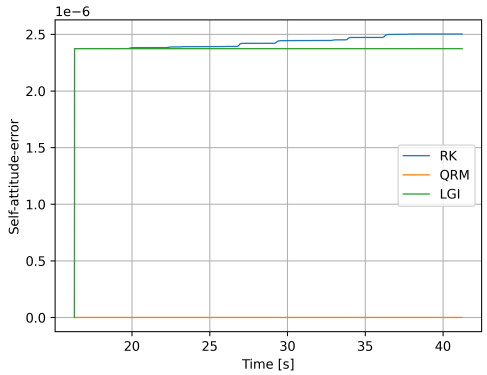
Figure 6: Angular velocities and numerical comparisons using the lemniscate trajectory.



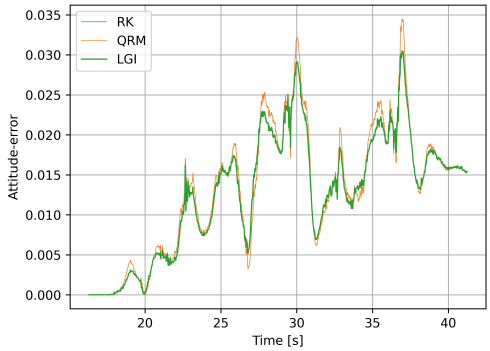
(a) Angular velocities of the race trajectory



(b) Determinant of the numerical solution $\boldsymbol{R}$

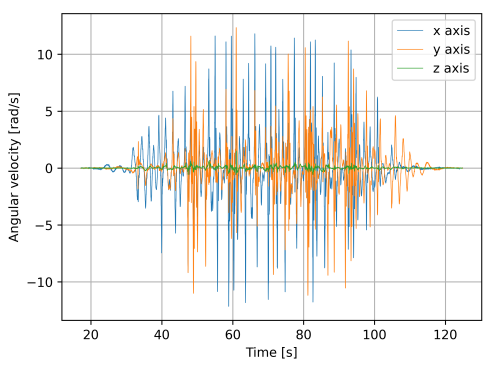


(c) Self-attitude error function $\Psi(\boldsymbol{R})$

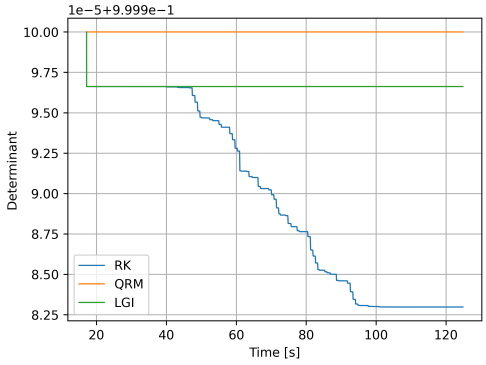


(d) Attitude error $\Psi(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}})$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

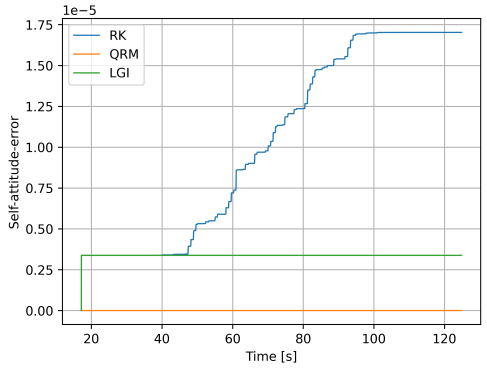
Figure 7: Angular velocities and numerical comparisons using the race trajectory.



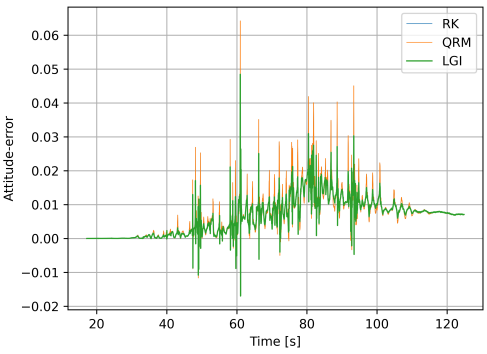
(a) Angular velocities of the random trajectory



(b) Determinant of the numerical solution $\boldsymbol{R}$



(c) Self-attitude error function $\Psi\left(\boldsymbol{R}\right)$

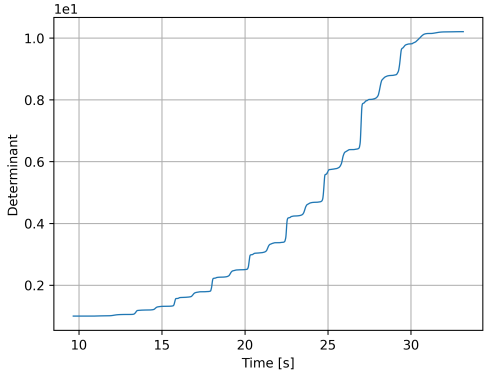


(d) Attitude error $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

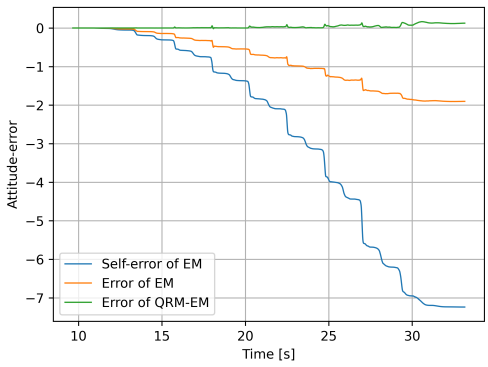
Figure 8: Angular velocities and numerical comparisons using the random trajectory.

Figure.6, 7, and 8 show that the Lie group integrator outperforms the other methods for the all three trajectories and has the highest numerical accuracy. Specifically, for the lemniscate and race trajectories, the three methods have comparable performances in preserving the two matrix structures, while the structure-preserving ability of the 4th-order Runge-Kutta method degenerates significantly for the random trajectory. It is also shown that the attitude error $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ of the Lie group integrator is significantly smaller than those of the other two methods when the angular velocities are large (see $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ at 27s in Fig.6, $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ at 37s in Fig.7, and $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ at 61s in Fig.8). This validates the effectiveness of the Lie group integrator method for challenging scenarios which are prevalent in mobile robots.

The numerical accuracies of the forward Euler method (labelled as 'EM') and the QR-factorization method based on the solution of the forward Euler method (labelled as 'QRM-EM') for the three kinds of trajectories, are shown in Fig.9, 10, and 11, respectively.

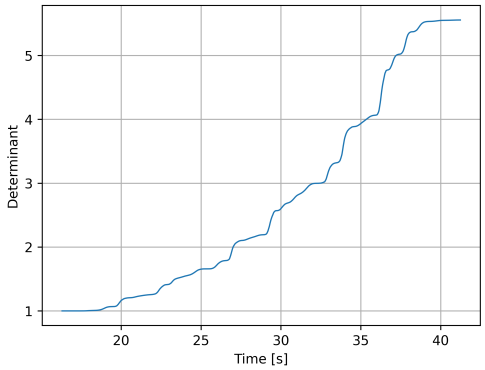


(a) Determinant of the numerical solution $\boldsymbol{R}$

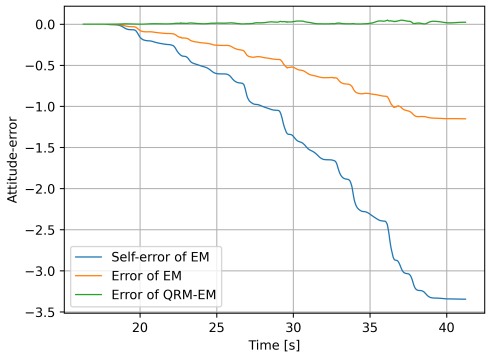


(b) Attitude error $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

Figure 9: Determinant and attitude error function of the EM method and the QRM-EM method for the lemniscate trajectory.

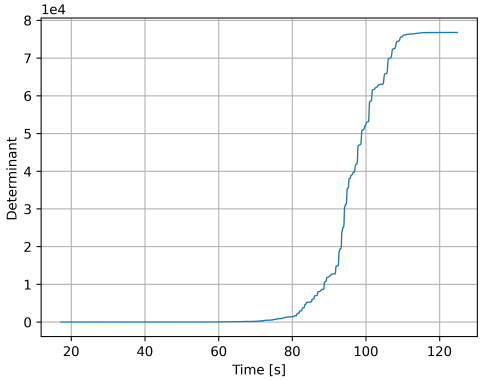


(a) Determinant of the numerical solution $\boldsymbol{R}$

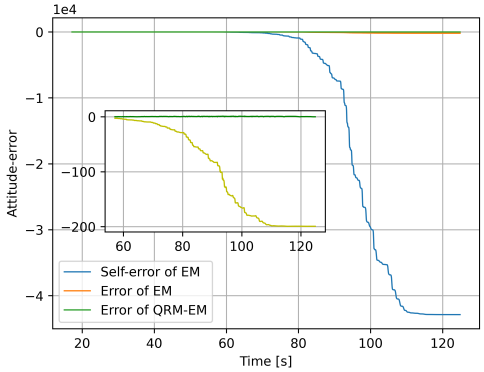


(b) Attitude error $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

Figure 10: Determinant and attitude error function of the EM method and the QRM-EM method for the race trajectory.



(a) Determinant of the numerical solution $\boldsymbol{R}$



(b) Attitude error $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ between $\boldsymbol{R}$ and $\boldsymbol{R}_{\mathrm{gt}}$

Figure 11: Determinant and attitude error function of the EM method and the QRM-EM method for the random trajectory.

As expected, the forward Euler method has the lowest numerical accuracy among these methods, and the corresponding determinant and attitude error even blow up for the random trajectory (see Fig.11). It is also evident that the QR-factorization method using the solution of the forward Euler method leads to a larger attitude error than the QR-factorization method using the solution of the 4th-order Runge-Kutta method, and this validates its dependence on the accuracy of the classical method used.

To better highlight the advantage of the Lie group integrator, the root mean square error (RMSE) was used to measure the attitude errors $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ of these methods for the three trajectories, as summarized in Table.1. It is shown that the Lie group integrator is able to generate the smallest RMSE among these methods for the all trajectories. Another observation worth noticing is that the 4th-order Runge-Kutta method has the comparable RMSE to the Lie group integrator for the lemniscate and race trajectories, meaning that the higher-order Runge-Kutta method may approximate the Lie group integrator well in some cases.

Trajectory	EM	RK	QRM	QRM-EM	LGI
Lemniscate	1.09480	0.01176	0.01175	0.05748	**0.01078**
Race	0.64289	0.01552	0.01551	0.01903	**0.01489**
Random	102.63197	0.00922	0.00922	0.46356	**0.00842**

Table 1: RMSE of $\Psi\left(\boldsymbol{R}, \boldsymbol{R}_{\mathrm{gt}}\right)$ of the five methods for the three kinds of trajectories

6 Conclusion

This term paper has studied the numerical integration methods for rotational kinematics in the form of a matrix ODE. Four methods were compared with each other, namely, the forward Euler method, the 4th-order Runge-Kutta method, the QR-factorization-based method, and the Lie group integrator method. Among them, the last two methods can preserve the matrix structure of rotation matrices. The theoretical comparisons show that the Runge-Kutta method is an approximation of the Lie group integrator from the perspective of a Taylor series, and can improve the accuracy of the approximation by increasing its order. The numerical comparisons using the real flight data validate the advantage of the Lie group integrator over the other methods. It is also shown that the higher-order Runge-Kutta method can approximate the Lie group integrator well in some cases, and the numerical accuracy of the QR-factorization-based method depends on the accuracy of the classical method used.

7 References

[1] M. Neunert, C. De Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1398–1404.

[2] J. C. Butcher, "A history of runge-kutta methods," *Applied numerical mathematics*, vol. 20, no. 3, pp. 247–260, 1996.

[3] D. Lewis and N. Nigam, "Geometric integration on spheres and some interesting applications," *Journal of Computational and Applied Mathematics*, vol. 151, no. 1, pp. 141–170, 2003.

[4] P. E. Crouch and R. Grossman, "Numerical integration of ordinary differential equations on manifolds," *Journal of Nonlinear Science*, vol. 3, no. 1, pp. 1–33, 1993.

[5] H. Munthe-Kaas, "Runge-kutta methods on lie groups," *BIT Numerical Mathematics*, vol. 38, no. 1, pp. 92–111, 1998.

[6] H. Munthe Kaas, "High order runge-kutta methods on manifolds," *Applied Numerical Mathematics*, vol. 29, no. 1, pp. 115–127, 1999.

[7] A. Iserles, H. Z. Munthe-Kaas, S. P. Nørsett, and A. Zanna, "Lie-group methods," *Acta numerica*, vol. 9, pp. 215–365, 2000.

[8] E. Celledoni, H. Marthinsen, and B. Owren, "An introduction to lie group integrators–basics, new developments and applications," *Journal of Computational Physics*, vol. 257, pp. 1040–1061, 2014.

[9] L. Jaulin, *Mobile robotics*. John Wiley & Sons, 2019.

[10] L. Bauersfeld, E. Kaufmann, P. Foehn, S. Sun, and D. Scaramuzza, "Neurobem: Hybrid aerodynamic quadrotor model," *arXiv preprint arXiv:2106.08015*, 2021.

[11] G. S. Chirikjian, *Stochastic Models, Information Theory, and Lie Groups, Volume 1: Classical Results and Geometric Methods*. Springer Science & Business Media, 2009.

[12] W. H. Enright, D. J. Higham, B. Owren, and P. W. Sharp, "A survey of the explicit runge-kutta method," 1995.

[13] Wikipedia, "Qr decomposition," 2021. [Online]. Available: https://en.wikipedia.org/wiki/QR_decomposition

[14] J. Schwichtenberg, "How is a lie algebra able to describe a group?" 2021. [Online]. Available: http://jakobschwichtenberg.com/lie-algebra-able-describe-group/

[15] cakey, "Prove that $so(3)$ acts transitively on the unit sphere s^2 of r^3," 2014. [Online]. Available: https://math.stackexchange.com/questions/573263/prove-that-so3-acts-transitively-on-the-unit-sphere-s2-of-bbb-r3

[16] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

8 Appendix

8.1 Proof of $\mathbf{RXR}^T = (\mathbf{Rx})^\wedge$

The proof is based on the orthogonality of rotation matrices: $\boldsymbol{R}\boldsymbol{R}^T = \mathbb{I}_3$. Given the rotation matrix written in the form of $\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3] \in \mathbb{R}^{3\times3}$, the orthogonality leads to

$$\boldsymbol{r}_1\boldsymbol{r}_1^T + \boldsymbol{r}_2\boldsymbol{r}_2^T + \boldsymbol{r}_3 r_3^T = \mathbb{I}_{3\times3} \tag{55}$$

These column vectors $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$ also satisfy the following equations, $\boldsymbol{r}_1^\wedge \boldsymbol{r}_2 = \boldsymbol{r}_3$, $\boldsymbol{r}_3^\wedge \boldsymbol{r}_1 = \boldsymbol{r}_2$, and $\boldsymbol{r}_2^\wedge \boldsymbol{r}_3 = \boldsymbol{r}_1$. Given the vector $\boldsymbol{x} = [x_1, x_2, x_3]^T \in \mathbb{R}^3$ as the dual vector of the skew-symmetric matrix $\boldsymbol{X} \doteq \boldsymbol{x}^\wedge$, the left hand side can be written as

$$\boldsymbol{R}\boldsymbol{X}\boldsymbol{R}^T = \left(\boldsymbol{r}_3\boldsymbol{r}_2^T - \boldsymbol{r}_2\boldsymbol{r}_3^T\right) x_1 + \left(\boldsymbol{r}_1\boldsymbol{r}_3^T - \boldsymbol{r}_3\boldsymbol{r}_1^T\right) x_2 + \left(\boldsymbol{r}_2\boldsymbol{r}_1^T - \boldsymbol{r}_1\boldsymbol{r}_2^T\right) x_3 \tag{56}$$

Using the above properties, we can simplify Eq.(56) as

$$\boldsymbol{R}\boldsymbol{X}\boldsymbol{R}^T = \boldsymbol{r}_1^\wedge \left(\mathbb{I}_{3\times3} - \boldsymbol{r}_1\boldsymbol{r}_1^T\right)\boldsymbol{x}_1 + \boldsymbol{r}_2^\wedge \left(\mathbb{I}_{3\times3} - \boldsymbol{r}_2\boldsymbol{r}_2^T\right)\boldsymbol{x}_2 + \boldsymbol{r}_3^\wedge \left(\mathbb{I}_{3\times3} - \boldsymbol{r}_3 r_3^T\right) x_3 \tag{57}$$

Since $\boldsymbol{r}_i^\wedge \boldsymbol{r}_i = 0,\ i = 1, 2, 3$, Eq.(57) reduces to

$$\begin{aligned} \boldsymbol{R}\boldsymbol{X}\boldsymbol{R}^T &= \boldsymbol{r}_1^\wedge x_1 + \boldsymbol{r}_2^\wedge x_2 + \boldsymbol{r}_3^\wedge x_3 \\ &= \left([\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \right)^\wedge \\ &= (\boldsymbol{R}\boldsymbol{x})^\wedge \end{aligned} \tag{58}$$

This completes the proof.

8.2 Python Code

Attached is the code that defines the four numerical integration algorithms

```
from casadi import *
class numerical_method:
    def __init__(self, dt_sample):
        self.h = dt_sample
        self.w = SX.sym('w', 3, 1)
        self.R = SX.sym('R', 3, 3)
    def skew_symmetric(self, v):
        v_hat = vertcat(
            horzcat(0, -v[2,0], v[1,0]),
             horzcat(v[2,0], 0, -v[0,0]),
             horzcat(-v[1,0], v[0,0], 0)
         )
         return v_hat
     def rotation_kinematics(self):
         dR = mtimes(self.R, self.skew_symmetric(self.w))
         self.dR_fn = Function('dR',[self.R, self.w], [dR], ['R_k', 'w_k'], ['dRf'])
     def QR_factorization(self):
         # Use Gram-Schmidt process to implement QR factorization
         r1, r2, r3 = self.R[:, 0], self.R[:, 1], self.R[:, 2]
```

```
20                  U1 = r1
21                  e1 = U1/norm_2(U1)
22                  U2 = r2 - dot(r2,e1)*e1
23                  e2 = U2/norm_2(U2)
24                  U3 = r3 - dot(r3,e1)*e1 - dot(r3,e2)*e2
25                  e3 = U3/norm_2(U3)
26                  Q = horzcat(e1, e2, e3)
27                  R = vertcat(
28                      horzcat(dot(r1,e1), dot(r2,e1), dot(r3,e1)),
29                      horzcat(0, dot(r2,e2), dot(r3,e2)),
30                      horzcat(0, 0, dot(r3,e3))
31                  )
32                  self.Q_fn = Function('Q', [self.R], [Q], ['R_k'], ['Qf'])
33                  self.R_fn = Function('R', [self.R], [R], ['R_k'], ['Rf'])
33              def matrix_exp(self):
34                  w_hat = self.skew_symmetric(self.w)
35                  what_exp = np.identity(3) +
36                 sin(self.h*norm_2(self.w))/(self.h*norm_2(self.w)) * self.h*w_hat +
37                  (1 - cos(self.h*norm_2(self.w)))/(self.h**2 * norm_2(self.w)**2) *
38                      self.h**2 * mtimes(w_hat, w_hat)
39                  self.whexp_fn = Function('what_exp', [self.w], [what_exp], ['w_k'], ['what_expf'])
40              def Euler_method(self, Rk, wk):
41                  k1 = self.dR_fn(R_k=Rk, w_k=wk)['dRf'].full()
42                  Rk1 = Rk + self.h * k1
43                  return Rk1
44              def Runge_Kutta_4th(self, Rk, wk):
45                  k1 = self.dR_fn(R_k=Rk, w_k=wk)['dRf'].full()
46                  k2 = self.dR_fn(R_k=Rk+self.h/2*k1, w_k=wk)['dRf'].full()
47                  k3 = self.dR_fn(R_k=Rk+self.h/2*k2, w_k=wk)['dRf'].full()
48                  k4 = self.dR_fn(R_k=Rk+self.h*k3, w_k=wk)['dRf'].full()
49                  Rk1 = Rk + self.h/6 * (k1 + 2*k2 + 2*k3 + k4)
50                  return Rk1
51              def QR_based(self, Rk1):
52                  Q = self.Q_fn(R_k=Rk1)['Qf'].full()
53                  R = self.R_fn(R_k=Rk1)['Rf'].full()
54                  Q = np.reshape(Q, (3, 3))
55                  R = np.reshape(R, (3, 3))
56                  v = np.diag(np.sign(R))
57                  Rk1_qr = np.matmul(Q, np.diag(v))
58                  return Rk1_qr
59              def Lie_group_integrator(self, Rk, wk, wk1):
60                  w_avg = (wk + wk1)/2
61                  exp_w = self.whexp_fn(w_k=w_avg)['what_expf'].full()
62                  exp_w = np.reshape(exp_w, (3, 3))
63                  Rk1_gi= np.matmul(Rk, exp_w)
64                  return Rk1_gi
```

The complete code running the simulation can be found at my `github` account through the link https://github.com/BinghengNUS/ME5701.