The Complex-Step Derivative Approximation on Matrix Lie Groups

Charles Champagne Cossette\textsuperscript{1}, Alex Walsh\textsuperscript{2}, and James Richard Forbes\textsuperscript{3}

Abstract—The complex-step derivative approximation is a numerical differentiation technique that can achieve analytical accuracy, to machine precision, with a single function evaluation. In this paper, the complex-step derivative approximation is extended to be compatible with elements of matrix Lie groups. As with the standard complex-step derivative, the method is still able to achieve analytical accuracy, up to machine precision, with a single function evaluation. Compared to a central-difference scheme, the proposed complex-step approach is shown to have superior accuracy. The approach is applied to two different pose estimation problems, and is able to recover the same results as an analytical method when available.

Index Terms—optimization and optimal control, localization

I. INTRODUCTION

ATTITUDE and pose, ubiquitous entities of interest in robotics problems, are most naturally represented as elements of matrix Lie groups. Path planning, state estimation, and control algorithms often require Jacobian computations with respect to attitude and pose. Often these Jacobians are computed analytically, by hand, via a Taylor-series expansion while adhering to the matrix Lie group structure of the problem \cite{1}. However, in some cases analytical computation of Jacobians may be impractical, necessitating a numerical procedure. Numerical computation of Jacobians is also useful for quickly comparing algorithms that require Jacobians, before investing effort into one specific algorithm and the associated analytically derived Jacobians. Numerical Jacobians can also be used to verify Jacobians that are derived by hand.

A variety of numerical differentiation techniques appropriate for matrix Lie groups can be found in the literature. In \cite{2} a forward-difference method is described for general matrix manifolds, a method that is used in the open-source software MANOPT \cite{5}. A central-difference method is employed in the open-source software GTSAM \cite{4}, and algorithmic differentiation methods are presented in \cite{5,6}. The Python-based software PYMANOPT \cite{7} is an open-source optimization toolbox for matrix manifolds that employs algorithmic differentiation. SOPHUS \cite{8} is another open-source C++ package that exploits the automatic differentiation functionality available in CERES \cite{9}, a nonlinear least-squares library developed by Google. However, algorithmic differentiation can be time consuming to implement and finite-differencing is prone to subtractive cancellation errors, thus limiting precision \cite{10}. The complex-step derivative approximation is a numerical method for computing first derivatives that does not suffer from subtractive cancellation errors \cite{10}. One of the earlier appearances of the complex-step derivative can be found in \cite{11}, where the derivatives of scalar functions of real variables are evaluated. In \cite{10}, the complex-step derivative is investigated further, along with its use in Fortran, C/C++, and other languages. An application to a multidisciplinary design optimization problem is also shown. This method has gained popularity due to its ability to realize machine-precision accuracy of derivative computations, and doing so without tuning the step size, since it can be reduced to an arbitrarily small value. The complex-step derivative also requires only one complex function evaluation, which is beneficial compared to central-differencing when the function is expensive to evaluate. The complex-step derivative is straight-forward to implement, especially in MATLAB, where the default variable type is complex.

This paper considers the formulation and application of the complex-step derivative approximation to functions of matrix Lie group elements. The aforementioned advantages of the standard complex-step derivative remain present, while the proposed method can be used to compute both left and right Jacobians. Various examples are presented, demonstrating the utility and advantages of the matrix Lie group version of the complex-step derivative. In particular, pose estimation problems are considered, one using the ETH Zürich EuRoC dataset \cite{12}, where analytical Jacobians are available for comparison, and one using the ‘Lost in the Woods’ dataset \cite{13}, where computation of analytical Jacobians is possible, but time consuming. When solving for the maximum a posteriori (MAP) estimate of the pose using a Gauss-Newton algorithm, it is shown that computing the Jacobians using the complex-step derivative realizes the same accuracy and convergence properties as when analytical Jacobians are used.

II. PRELIMINARIES

A. Matrix Lie Groups

A matrix Lie group \( \mathcal{G} \) is a Lie group that consists of the set of \( m \times m \) invertible matrices, where the group operation is matrix multiplication \cite{14} Ch. 10.2. From the definition of a group, a matrix Lie group is closed under matrix multiplication. That is, given \( X, Y \in \mathcal{G} \), it follows that \( XY \in \mathcal{G} \). A matrix Lie group is a closed subgroup of the general linear group defined by \cite{15} Ch. 1.1

\[ GL(m, \mathbb{C}) = \{ X \in \mathbb{C}^{m \times m} \mid \det(X) \neq 0 \} \]
which is also a matrix Lie group. The matrix Lie algebra of \( G \) is denoted \( \mathfrak{g} \), and is defined as [15, Ch. 3.3],
\[
\mathfrak{g} = \{ \Xi \mid \exp(\Xi) \in G, \forall t \in \mathbb{R} \}. \tag{1}
\]
It can be shown that the matrix Lie algebra defined by (1) is a valid Lie algebra [15, Ch. 3.1], and is a vector space closed under the operation of the Lie bracket \([\cdot, \cdot]\), which can be computed by \([A, B] = AB - BA \in \mathfrak{g}\) for all \( A, B \in \mathfrak{g} \).

The wedge operator \((\cdot)\wedge : \mathbb{R}^n \rightarrow \mathfrak{g}\) maps a column matrix to the matrix Lie algebra. The exponential map \(\exp(\cdot) : \mathfrak{g} \rightarrow G\) maps an element of the matrix Lie algebra to the matrix Lie group, and is computed using the matrix exponential. The only maps an element of the matrix Lie algebra to the matrix Lie group, and is computed using the matrix exponential. A parameterization of the group \(G\) can be retrieved from \(X\) via
\[
\xi = \ln(\exp(\zeta^\wedge)),
\]
where \(\zeta \in \mathbb{R}^n\). The “vee” operator \((\cdot)^\vee : \mathfrak{g} \rightarrow \mathbb{R}^n\) maps an element of the matrix Lie algebra to a column matrix. The logarithmic map \(\ln(\cdot) : G \rightarrow \mathfrak{g}\) maps an element of the matrix Lie group to the matrix Lie algebra, and is computed by the matrix logarithm. A parameterization of the group \(G\) can be retrieved from \(X\) via
\[
\xi = \ln(\exp(\Xi^\wedge)),
\]
when the matrix logarithm is well defined. The adjoint representation of \(X\) is denoted \(\text{Ad}(X)\), such that \((\text{Ad}(X)\zeta^\wedge)^\wedge = X\zeta^\wedge X^{-1}\), \(\zeta \in \mathbb{R}^n\). This leads to the identity
\[
\exp((\text{Ad}(X)\zeta^\wedge)^\wedge) = X \exp(\zeta^\wedge)X^{-1}.
\]
The Baker-Campbell-Hausdorff (BCH) formula is the solution to
\[
z = \ln(\exp(\xi^\vee) \exp(\xi_2^\vee)),
\]
and the exact solution is an infinite sum [1]. A first-order approximation to the BCH formula is
\[
\ln(\exp(\xi^\vee) \exp(\xi_2^\vee)) = \xi^\vee_1 + \xi^\vee_2,
\]
which is exact in the event that \(\xi^\vee_1, \xi^\vee_2 = 0\). Such an approximation is typically used when both \(\xi_1\) and \(\xi_2\) are assumed to be small. The details of the special Euclidean groups \(SE(2)\), \(SE(3)\), and the group of double direct isometries \(SE_2(3)\) can be found in the appendix.

### B. Gauss-Newton Algorithm

The Gauss-Newton algorithm is an optimization algorithm appropriate for nonlinear least-squares functions of the form
\[
J(x) = \frac{1}{2} e(x)^T W e(x), \tag{2}
\]
where \(W \in \mathbb{R}^{q \times q}\) is a symmetric positive definite weight matrix and \(e : \mathbb{R}^p \rightarrow \mathbb{R}^q\) is some error function. Employing Newton’s method directly on (2) requires the Hessian of \(J(x)\), which is potentially difficult to obtain. An alternate strategy is to substitute a first-order approximation of \(e(x)\) about some nominal \(\bar{x}\), given by [1] Ch. 4.3
\[
e(\bar{x} + \delta x) \approx e(\bar{x}) + \left. \frac{\partial e(x)}{\partial x} \right|_{x=\bar{x}} \delta x,
\]
into (2), thus yielding the Jacobian and a Hessian approximation of \(J(x)\),
\[
\begin{align*}
J(x) & \approx \frac{1}{2} e(\bar{x})^T W e(\bar{x}) \\
& + e(\bar{x})^T W \left. \frac{\partial e(x)}{\partial x} \right|_{x=\bar{x}} + \frac{1}{2} \left. \frac{\partial^2 e(x)}{\partial x^2} \right|_{x=\bar{x}} W \left. \frac{\partial e(x)}{\partial x} \right|_{x=\bar{x}}^T \delta x.
\end{align*}
\]
The Gauss-Newton algorithm then proceeds identically to Newton’s method, where the nominal point is iterated by \(\bar{x}_t = \bar{x}_{t-1} + \delta x_{t-1}\). The step \(\delta x_{t-1}\) is calculated as
\[
\delta x_{t-1} = - \left( \left. \frac{\partial J(x)}{\partial \delta x} \right|_{x=\bar{x}_{t-1}} \right)^{-1} \left. \frac{\partial J(x)}{\partial x} \right|_{x=\bar{x}_{t-1}}^T.
\]

### III. THE COMPLEX-STEP DERIVATIVE APPROXIMATION

#### A. Review

Consider the complex-differentiable function \(f : \mathbb{C} \rightarrow \mathbb{C}\) perturbed about the nominal point \(\bar{x}\) by \(jh\) where \(\bar{x}, h \in \mathbb{R}\) and \(j = \sqrt{-1}\). A Taylor series expansion yields
\[
\begin{align*}
f(\bar{x} + jh) &= f(\bar{x}) + \left. \frac{\partial f(z)}{\partial z} \right|_{z=\bar{x}} jh \\
&- \frac{1}{2} \left. \frac{\partial^2 f(z)}{\partial z^2} \right|_{z=\bar{x}} h^2 + \frac{1}{3!} \left. \frac{\partial^3 f(z)}{\partial z^3} \right|_{z=\bar{x}} jh^3 + \ldots \tag{3}
\end{align*}
\]
If \(f(\bar{x})\) is assumed to be real for all real \(\bar{x}\), then, to first order, taking the imaginary portion of (3) yields [11]
\[
\left. \frac{\partial f(z)}{\partial z} \right|_{z=\bar{x}} = \frac{\text{Im}\{f(\bar{x} + jh)\}}{h} + O(h^2).
\]
This is valid as long as \(f(\bar{x}) \in \mathbb{R}\) for all \(\bar{x} \in \mathbb{R}\), and that derivatives are evaluated at strictly real nominal points. From a practical standpoint, a user is often attempting to find derivatives of \(f : \mathbb{R} \rightarrow \mathbb{R}\). Providing that this can be extended to \(f : \mathbb{C} \rightarrow \mathbb{C}\) such that \(f\) is complex-differentiable, then with a minor abuse of notation, this can construct a derivative approximation for \(f(x)\) as written in [10] [11].
\[
\left. \frac{\partial f(x)}{\partial x} \right|_{x=\bar{x}} \approx \frac{\text{Im}\{f(x + jh)\}}{h}.
\]
Since there are no subtractive cancellation errors, the complex-step derivative approximation can produce machine-precision approximations by reducing \(h\) to an arbitrarily small step size.

#### B. The Complex-Step Derivative on Matrix Lie Groups

Consider a complex-differentiable function \(f : G \rightarrow \mathbb{C}\) where \(G \subset GL(m, \mathbb{C})\), \(X(e^R) = X \exp(e^{R^T})\) is parametrizable by a perturbation \(e^R = [e^R_1, e^R_2, \ldots e^R_n]^T \in \mathbb{C}^n\) on the right, and \(X \in \mathbb{R}^{m \times m}\) is some nominal value of \(X\). Consider perturbing \(f(X(e^R))\) by \(e^R = 0 + jh1_i\), where \(1_i\) is the \(i\)th column of the appropriately-dimensioned identity matrix \(1\). The composition \(f(X(e^R))\) has essentially recast \(f\) as
To this end, it is achievable with a sufficient reduction in step size. The right Jacobian can identically be obtained by instead parametrizing $\hat{\mathbf{X}}$ individually computing the derivatives using (5) with $T=1$. The left Jacobian can be determined analytically using the first-order approximation where

$$f(\hat{\mathbf{X}} \exp((jh\hat{\mathbf{1}})^\wedge)) = f(\hat{\mathbf{X}}) + \frac{\partial f(\mathbf{X}(e^R))}{\partial e^R} \bigg|_{e^R = 0} jh - \frac{1}{2} \frac{\partial^2 f(\mathbf{X}(e^R))}{\partial e^R \partial e^R} \bigg|_{e^R = 0} h^2 + \mathcal{O}(h^3).$$

(4)

Since it is assumed that $f(\hat{\mathbf{X}}) \in \mathbb{R}$, taking the imaginary component of (4) yields an approximation for the derivative

$$\frac{\partial f(\mathbf{X}(e^R))}{\partial e^R} \approx \frac{\text{Im} \{ f(\mathbf{X} \exp((jh\hat{\mathbf{1}})^\wedge)) \}}{h}.$$

(5)

The right Jacobian $\frac{\partial f(\mathbf{X}(e^R))}{\partial e^R}$ can be obtained by individually computing the derivatives using (5) with $i = 1, 2, \ldots, n$. The left Jacobian can identically be obtained by instead parametrizing $\mathbf{X}$ with $\mathbf{X}(e^L) = \exp(e^L)\mathbf{X}$. This leads to

$$\frac{\partial f(\mathbf{X}(e^L))}{\partial e^L} \approx \frac{\text{Im} \{ f(\exp((jh\hat{\mathbf{1}})^\wedge))\mathbf{X} \}}{h}.$$

(6)

Note that the superscripts on $e^R$ and $e^L$ are simply labels that correspond to right and left perturbations, respectively, as opposed to exponents.

**Example 1:** Consider the function

$$f(\mathbf{T}) = \mathbf{v}^T \mathbf{T} \mathbf{y},$$

where $\mathbf{T} \in SE(3)$ and $\mathbf{v}, \mathbf{y} \in \mathbb{R}^4$. The left Jacobian can be determined analytically using the first-order approximation $\mathbf{T} = \exp(e^L)\mathbf{T} \approx (1 + e^L)\mathbf{T}$ and a Taylor series expansion. To this end,

$$f(\exp(e^L)\mathbf{T}) = \mathbf{v}^T \exp(e^L)\mathbf{T} \mathbf{y} \approx \mathbf{v}^T(1 + e^L)\mathbf{T} \mathbf{y} = \mathbf{v}^T \mathbf{T} \mathbf{y} + \mathbf{v}^T(\mathbf{T} \mathbf{y})^\circ e^L,$$

(7)

where the $(\cdot)^\circ$ operator is defined in the appendix. The elements of $\partial f(\mathbf{T}(e^L))/\partial e^L$ are computed using (6) with varying step sizes $h$, and the results are compared with a central-difference scheme in Fig. [1]. The error is computed by taking the relative 2-norm of the difference between the analytical and numerical solutions. Like the standard complex-step derivative, the complex-step derivative tailored to the matrix Lie group $SE(3)$ is able to achieve analytic accuracy, up to machine precision, for small enough $h$, while the central-difference derivative is not.

**Example 2:** Consider the nonlinear least-squares function

$$J(\mathbf{T}) = \frac{1}{2} \mathbf{e}(\mathbf{T})^T \mathbf{W} \mathbf{e}(\mathbf{T}),$$

(8)

where $\mathbf{T} = \exp(e^L)\mathbf{T} \in SE(3)$, $\mathbf{W} \in \mathbb{R}^{6 \times 6}$ is a symmetric positive definite weight matrix, and the error is given by

$$\mathbf{e}(\mathbf{T}) = \ln(\mathbf{T}^{-1} \mathbf{T}^\text{ref})^\vee.$$

The matrix $\mathbf{T}^\text{ref} \in SE(3)$ is some reference point used to construct the error. The Jacobian $\partial \mathbf{e}(\mathbf{T}(e^L))/\partial e^L$ can be used to construct Jacobian and Hessian approximations of $J(x)$, which are used in the Gauss-Newton algorithm. Like in Example 1, the analytical left Jacobian can be determined by perturbing $\mathbf{T}$ on the left,

$$\mathbf{e}(\exp(e^L)\mathbf{T}) = \ln(\mathbf{T}^{-1} \exp(-e^L)\mathbf{T}^\text{ref})^\vee = \ln(\exp((-\text{Ad}(\mathbf{T}^{-1}) e^L)^\wedge) \mathbf{T}^{-1} \mathbf{T}^\text{ref})^\vee \approx \mathbf{e}(\mathbf{T}) + (-\text{Ad}(\mathbf{T}^{-1}) e^L)^\wedge,$$

(9)

where, in the last line, a first-order approximation to the BCH formula has been used.

The elements of the Jacobian $\partial \mathbf{e}(\mathbf{T}(e^L))/\partial e^L$ were also calculated using (6) with a step size of $h = 10^{-20}$. An
optimization was performed with both Jacobian calculation methods, where the Gauss-Newton step $\delta \epsilon_{\ell-1}$ is determined from

$$
\delta \epsilon_{\ell-1} = \left[ \left( \frac{\partial \epsilon}{\partial \epsilon^l} \right)^T W \left( \frac{\partial \epsilon}{\partial \epsilon^l} \right) \right]^{-1} \left[ - \left( \frac{\partial \epsilon}{\partial \epsilon^l} \right)^T W \epsilon(\mathbf{T}) \right],
$$

and the argument of $\epsilon(\mathbf{T}(\epsilon^l))$ is dropped for conciseness. The point is updated by

$$
\mathbf{T}_\ell = \exp(\delta \epsilon_{\ell-1})\mathbf{T}_{\ell-1}.
$$

As shown in Fig. 3 using both an analytic Jacobian or a complex-step Jacobian results in an optimum being reached in a single step. Note that calculating Jacobians using the complex-step is shown to have a minor improvement in cost function reduction as compared to the analytical method. The reason is that the analytical method uses a first-order BCH function reduction as compared to the analytical method. The approximation, which is ultimately slightly less accurate than the machine-precision complex-step Jacobian calculations.

IV. Batch Estimation

The methodology of Example 2 is now applied to a practical state estimation problem. Consider the task of estimating the position and attitude of a rigid body at different points in time $t_0, t_1, \ldots, t_K$ using various measurements. The state of the rigid body at a discrete point in time $t_k$ can be represented by the matrix Lie group element $\mathbf{T}_k \in \mathcal{G}$, where $\mathcal{G}$ will depend on the estimation task.

A. Maximum A Posteriori Estimation

The MAP approach [1, Ch. 8.2.5] to estimate the states in a batch framework results in the minimization of the least-squares cost function shown in (8), where the errors to be minimized are

$$
\mathbf{e}(\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_K) =
\begin{bmatrix}
\mathbf{e}_{u,0} \\
\mathbf{e}_{u,1} \\
\vdots \\
\mathbf{e}_{u,K} \\
\mathbf{e}_{y,0} \\
\vdots \\
\mathbf{e}_{y,K}
\end{bmatrix}.
$$

The error term $\mathbf{e}_{u,0}$ represents an error between the known initial state $\mathbf{T}_0$, with uncertainty, and the estimated initial state $\mathbf{T}_0$. This term is computed as

$$
\mathbf{e}_{u,0} = \ln(\mathbf{T}_0^{-1}\mathbf{T}_0)^\vee.
$$

The process error terms $\mathbf{e}_{u,1}, \ldots, \mathbf{e}_{u,K}$ are a function of a discrete-time process model of the form $\mathbf{T}_k = \mathbf{F}(\mathbf{T}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1})$ where $\mathbf{u}_{k-1}$ and $\mathbf{w}_{k-1}$ are the input and zero-mean process noise at time $t_{k-1}$, respectively. These error terms are calculated as

$$
\mathbf{e}_{u,k} = \ln(\mathbf{T}_k^{-1}\mathbf{F}(\mathbf{T}_{k-1}, \mathbf{u}_{k-1}, 0)^\vee).
$$

Finally, the terms $\mathbf{e}_{y,0}, \ldots, \mathbf{e}_{y,K}$ correspond to the errors between measurements, and a measurement model of the form $\mathbf{y}_k = \mathbf{g}(\mathbf{T}_k, \mathbf{\nu}_k)$, where $\mathbf{\nu}_k$ is zero-mean measurement noise. Hence, the measurement errors are

$$
\mathbf{e}_{y,k} = \mathbf{y}_k - \mathbf{g}(\mathbf{T}_k, 0).
$$

Following the MAP formulation the weight in (8) is

$$
\mathbf{W} = \text{diag}(\mathbf{P}_0^{-1}, \mathbf{Q}_1^{-1}, \ldots, \mathbf{Q}_K^{-1}, \mathbf{R}_0^{-1}, \ldots, \mathbf{R}_K^{-1}),
$$

where the matrix $\mathbf{P}_0$ is a covariance matrix associated with the uncertainty in the initial state, $\mathbf{T}_0$. The matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$ are covariance matrices associated with the process and measurement noises, respectively.

The goal is to find $\mathbf{T}_0, \ldots, \mathbf{T}_K$ that minimize the least-squares cost function given by (8). To use a Gauss-Newton algorithm, the right (or left) Jacobian $\partial \mathbf{e}(\mathbf{T}(\epsilon^R)) / \partial \epsilon^R$ is needed, where $\epsilon^R = [\epsilon_0^R \ldots \epsilon_K^R]^T$ is a matrix that consists of perturbations to the individual estimated states. Since the error $\mathbf{e}(\mathbf{T}_0, \ldots, \mathbf{T}_K)$ is a function of $K$ different Lie group elements, it is worth mentioning a simple technique that allows a user to treat the same function as a function of a single matrix Lie group element, as shown next in Section IV-B. The Jacobians can then be computed using (5) or (6).

B. Recasting $f(\mathbf{X}_0, \ldots, \mathbf{X}_K)$ as $f(\mathbf{X})$

Consider a function $f(\mathbf{X}_0, \ldots, \mathbf{X}_K) \in \mathbb{R}$ where $\mathbf{X}_0, \ldots, \mathbf{X}_K \in \mathcal{G}$. Let $\mathbf{X}_i = \mathbf{X}_i \exp(\epsilon_i^R)$. Define

![Fig. 3. Trajectory visualization of a batch-estimation solution from the EuRoC Dataset.](image-url)
Virtually identical performance is achieved to the analytical solution, and of functions of multiple matrix Lie group elements will be consolidated under $T$, as described in Section IV-B The right Jacobian is

$$\frac{\partial \mathbf{e}(T)}{\partial \epsilon^R} \approx \begin{bmatrix} -1 & \mathbf{F}_0 & \mathbf{F}_K \\ \mathbf{H}_0 & \mathbf{H}_K \end{bmatrix}$$

where

$$\mathbf{F}_k = \text{Ad}(\mathbf{T}_{k-1}^{-1})\mathbf{F}_{k-1}^{\text{op}},$$

$$\mathbf{F}_{k-1}^{\text{op}} = \begin{bmatrix} \mathbf{C}_{k-1} & \mathbf{v}_{k-1} + \mathbf{T}\mathbf{g} & \mathbf{r}_{k-1} + \mathbf{T}\mathbf{v}_{k-1} \\ 1 & 0 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{T}_{k} \mathbf{p}^\odot,$$

where $T = t_k - t_{k-1}$, $\mathbf{p} = [0 \ 1]^T$, and $\mathbf{g}$ is the gravity vector resolved in the datum frame. These expressions require first-order approximations to the BCH formula, similar to Example 2. This is common procedure, as the approximation becomes more accurate as errors become small [1] [16].

A Gauss-Newton optimization is performed on the MH_03_medium dataset. For simplicity, the accelerometer and gyroscope measurements are downsampled from the original 200 Hz in order to reduce the amount of variables in the optimization procedure. An alternative to downsampling is to perform IMU preintegration as described in [17], but this is beyond the scope of this paper. The specifications of the
The initial state, $\mathbf{T}_0$, is set to the ground truth, and hence the diagonal of $\mathbf{P}_0$ is given arbitrarily small numbers. The matrix $\mathbf{Q}_k$ was further tuned to yield better performance, after obtaining the nominal noise values provided in the EuRoC dataset. Using the initial state, the process model is directly integrated using the accelerometer and gyroscope measurements, which then provides an initial guess for the poses at all the discrete time points. This dead reckoning solution is then used to initialize the Gauss-Newton algorithm.

Figure 5 shows a visualization of the trajectory once the optimization procedure has converged. Figure 4 shows the poses at all the discrete time points. This dead reckoning measurement, which then provides an initial guess for the process model is directly integrated using the accelerometer and gyroscope measurements, which then provides an initial guess for the poses at all the discrete time points. This dead reckoning solution is then used to initialize the Gauss-Newton algorithm.

In this example, BCH approximations in the analytical Jacobians did not create any difference in the convergence history since the errors are initialized to be small in the the dead reckoning step. A central-difference scheme was also used to calculate Jacobians, and after multiple trial-and-error attempts with different step sizes, an identical convergence history to what is shown in Fig. 4 was observed. However, the central-difference method requires twice as many function evaluations as the complex-step method, and therefore required approximately twice the total computing time. Finally, Fig. 5 shows the 2-norm of the difference between the batch-estimation solution and the ground truth. The errors are small, indicating the MAP framework has converged close to the ground truth.

D. The ‘Lost in the Woods’ Dataset

The ‘Lost in the Woods’ dataset consists of a mobile wheeled robot navigating through a “forest” of tubes [13], as seen in Figure 6. The robot is equipped with wheel odometry providing forward velocity measurements, denoted $u_k^{vel}$, and angular velocity measurements, denoted $u_k^{ang}$. Furthermore, the robot has a laser range finder that provides range and bearing measurements to pre-identified landmarks (the tubes shown in Figure 6), denoted $r_k^\ell$, $\phi_k^\ell$ for landmark $\ell$ at $t_k$, respectively. The positions of the landmarks in a datum reference frame are

The process model consists of the nonholonomic vehicle kinematics. Written in the form

$$\mathbf{T}_k = \mathbf{F}(\mathbf{T}_{k-1}, \mathbf{u}_{k-1}, \mathbf{w}_{k-1}),$$

where

$$\mathbf{y}_k = \mathbf{g}(\mathbf{T}_k, \mathbf{v}_k),$$

and $\mathbf{v}_k$ is zero-mean normally distributed noise, $\mathbf{D} = [\mathbf{I} \ 0]$, $\mathbf{p} = [d \ 0 \ 1]^T$, and $d$ is the distance between the laser range finder and the reference point on the robot. Computed the Jacobians associated with the measurement model by hand is, although not impossible, laborious due to the $atan2(\cdot, \cdot)$ term. Hence, the complex-step derivative is used to directly evaluate the right Jacobian $\partial \mathbf{e}(\mathbf{T})/\partial \mathbf{e}^R$ for use in the Gauss-Newton optimization.

Dead reckoning was performed using wheel odometry in order to generate an initial guess for the Gauss-Newton optimization. All measurements were downsampled from the
original 10 Hz to 5 Hz in order to limit the number of variables in the optimization procedure. The $Q_k$ and $R_k$ matrices were directly formed from the discrete-time covariances provided in the dataset [13]. The initial state $T_0$ was set to be a random perturbation from the ground truth.

The algorithm converged in 6 iterations, and produced a trajectory visualizable in Figure [7]. The errors are shown in Figure [8] which show good performance when compared to the ground truth position and attitude data. This can also be achieved with central-difference, but again, the computation time is significantly longer, and the step size must be tuned.

V. CONCLUSION

This paper has shown that the complex-step derivative can successfully be used to obtain Jacobians of functions that have matrix Lie group elements as arguments. Machine-precision can be achieved with a single complex function evaluation. To use the complex-step, functions must be programmed to accept the complex-step, but again, the computation time is significantly longer, and the step size must be tuned. There is a multitude of other potential applications for this tool, such as numerical linearization of high-fidelity dynamics models, real-time state estimation and Kalman filtering [19], and the training of matrix Lie group-based neural networks [20]. For second derivatives, the complex-step is unfortunately unable to realize machine-precision accuracy. However, methods are available to improve the accuracy [21], which are likely extendible to matrix Lie groups. Furthermore, if an analytical Jacobian is known, the Hessian can be determined with machine precision using the complex-step [10].

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{trajectory.png}
\caption{2D trajectory trace for the ‘Lost in the Woods’ dataset. The solution using the complex-step derivative shows excellent agreement with the ground truth data.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{errors.png}
\caption{Error in position $x$, $y$ and attitude $\theta$ between estimated solution and ground truth (blue), along with $\pm 3$ standard deviation bounds (black). There is less than 10 cm of position error, and less than 0.1 rad of attitude error.}
\end{figure}

\section*{APPENDIX}

\subsection*{A. The Special Euclidean Group SE(2)}

The group $SE(2)$ is defined as [14],

$$SE(2) = \left\{ T = \begin{bmatrix} C & r \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{3 \times 3} \middle| C \in SO(2), \, r \in \mathbb{R}^2 \right\},$$

where $SO(n)$ refers to the Special Orthogonal Group consisting of orthonormal matrices with unit determinant. The matrix Lie algebra associated with $SE(2)$ is

$$se(2) = \{ \Xi = \xi^\phi \in \mathbb{R}^{3 \times 3} \mid \xi \in \mathbb{R}^3 \},$$

where

$$\xi^\phi = \begin{bmatrix} \xi^\phi_x \\ \xi^\phi_y \\ \xi^\phi_z \end{bmatrix}^\phi = \begin{bmatrix} 0 & -\xi^\phi_z & \xi^\phi_y \\ \xi^\phi_z & 0 & -\xi^\phi_x \\ -\xi^\phi_y & \xi^\phi_x & 0 \end{bmatrix}. $$

The closed-form expression for the exponential map $\exp : se(2) \rightarrow SE(2)$ is

$$\exp(\xi^\phi) = \begin{bmatrix} C & J_\ell \xi^\phi^T \\ 0 & 1 \end{bmatrix},$$

where $\xi^\phi = [\xi_1^\phi, \xi_2^\phi]^T$ and

$$J_\ell = \frac{1}{\xi^\phi} \begin{bmatrix} \sin(\xi^\phi) & -\cos(\xi^\phi) \\ (1 - \cos(\xi^\phi)) & \sin(\xi^\phi) \end{bmatrix}. $$

\subsection*{B. The Special Euclidean Group SE(3)}

The matrix Lie group $SE(3)$ is defined as [11, Ch. 7.1.2]

$$SE(3) = \left\{ T = \begin{bmatrix} C & r \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \middle| C \in SO(3), \, r \in \mathbb{R}^3 \right\}. $$

The matrix Lie algebra associated with $SE(3)$ is

$$se(3) = \{ \Xi = \xi^\phi \in \mathbb{R}^{4 \times 4} \mid \xi \in \mathbb{R}^6 \},$$

where

$$\xi^\phi = \begin{bmatrix} \xi^\phi_x \\ \xi^\phi_y \\ \xi^\phi_z \\ \xi^\phi_r \end{bmatrix}^\phi = \begin{bmatrix} \xi^\phi_x^* \\ \xi^\phi_y^* \\ \xi^\phi_z^* \\ \xi^\phi_r \end{bmatrix}, \quad \xi^\phi_x, \xi^\phi_r \in \mathbb{R}^3, $$

\text{E. Special Euclidean Group SE(3)}
and
\[ \xi^{\phi \times} = \begin{bmatrix} \xi^\phi \\ \xi^v \end{bmatrix} = \begin{bmatrix} 0 & -\xi^3 & \xi_2 \\
\xi_2 & 0 & -\xi_1 \\
-\xi_1 & 0 & 0 \end{bmatrix}. \]

The closed-form expression for the exponential map \( \exp : se(3) \rightarrow SE(3) \) is
\[ \exp(\xi^{\phi \times}) = \begin{bmatrix} \exp(\xi^\phi) & J_\xi \xi^v \\
0 & 1 \end{bmatrix}, \]
where
\[ J_\xi = \frac{\sin(\phi)}{\phi} \mathbf{1} + \left(1 - \frac{\sin(\phi)}{\phi}\right) \mathbf{a} a^\top + \frac{1 - \cos(\phi)}{\phi} \mathbf{a} a^\times, \]
\[ \exp(\xi^{\phi \times}) = \cos(\phi) \mathbf{1} + (1 - \cos(\phi)) \mathbf{a} a^\top + \sin(\phi) \mathbf{a} a^\times, \]
and \( \phi = \|\xi^\phi\| \) and \( \mathbf{a} = \xi^v / \phi \). The matrix \( J_\xi \) is known as the left Jacobian of the group \( SO(3) \). It is also useful to define the operator \([1] Ch. 7.1.8\]
\[ \mathbf{p}^\odot = \begin{bmatrix} \mathbf{e} \\
\eta_1 \\
\eta_2 \end{bmatrix} = \begin{bmatrix} -\mathbf{e}^\times & \eta_1 \mathbf{1} & \eta_2 \mathbf{1} \\
0 & 0 & 0 \\
0 & 0 & 0 \end{bmatrix}, \]
such that \( \mathbf{x}^\times \mathbf{p} = \mathbf{p}^\odot \mathbf{x} \) holds.

C. The Group of Double Direct Isometries \( SE_2(3) \)

The matrix Lie group \( SE_2(3) \) is defined as
\( SE_2(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{C} & \mathbf{v} & \mathbf{r} \\
0 & 1 & 0 \\
0 & 0 & 1 \end{bmatrix} \mid \mathbf{C} \in SO(3), \mathbf{v}, \mathbf{r} \in \mathbb{R}^3 \right\} \).

The matrix Lie algebra associated with \( SE_2(3) \) is
\( se_2(3) = \{ \Xi = \xi^\wedge \in \mathbb{R}^{5 \times 5} \mid \xi \in \mathbb{R}^9 \} \),
where
\[ \xi^\wedge = \begin{bmatrix} \xi^\phi \\
\xi^v \\
\xi^r \end{bmatrix}^\wedge = \begin{bmatrix} \xi^\phi \xi^v \\
0 & 0 & 0 \\
0 & 0 & 0 \end{bmatrix} \], \( \xi^\phi, \xi^v, \xi^r \in \mathbb{R}^3 \).

The closed-form expression for the exponential map \( \exp : se_2(3) \rightarrow SE_2(3) \) is
\[ \exp(\xi^\wedge) = \begin{bmatrix} \exp(\xi^\phi) & J_\xi \xi^v \\
0 & 1 \\
0 & 0 \end{bmatrix}, \]
It is also useful to define the operator
\[ \mathbf{p}^\odot = \begin{bmatrix} \mathbf{e} \\
\eta_1 \\
\eta_2 \end{bmatrix}^\odot = \begin{bmatrix} -\mathbf{e}^\times \eta_1 \mathbf{1} & \eta_2 \mathbf{1} \\
0 & 0 \\
0 & 0 \end{bmatrix}, \]
where \( \mathbf{e} \in \mathbb{R}^3 \) and \( \eta_1, \eta_2 \in \mathbb{R} \), such that \( \mathbf{x}^\times \mathbf{p} = \mathbf{p}^\odot \mathbf{x} \) holds.

ACKNOWLEDGMENT

The authors would like to thank Jonathan Arsenaunt and Thomas Hitchcox for their many helpful discussions.

REFERENCES

[1] T. Barfoot, State Estimation for Robotics. Toronto, ON: Cambridge University Press, 2019.
[2] P. Absil, R. Mahony, and R. Sepulchre, Optimization Algorithms on Matrix Manifolds. Princeton Uni, Press, 2008.
[3] N. Boumal, B. Mishra, P.-A Absil, and R. Sepulchre, “Manopt, a Matlab Toolbox for Optimization on Manifolds,” Machine Learning Research, vol. 15, pp. 1455–1459, 2014.
[4] GTSAM, Math of GTSAM, 2019. [Online]. Available: https://github.com/borglab/gtsam (visited on 09/03/2019).
[5] K. Røbenack, J. Winkler, and S. Wang, “LIEDRIVERS - A Toolbox for the Efficient Computation of Lie Derivatives Based on the Object-Oriented Algorithmic Differentiation Package ADOL-C,” in Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, ETH Zürich, 2011, pp. 57–66.
[6] H. Sommer, C. Pradalier, and P. Furgale, “Automatic Differentiation on Differentiable Manifolds as a Tool for Robotics,” in Robotics Research: The 16th International Symposium, 2013, pp. 505–520.
[7] J. Townsend and S. Weichwald, “Pymanopt: A Python Toolbox for Optimization on Manifolds Using Automatic Differentiation,” Journal of Machine Learning Research, vol. 17, no. 1, pp. 1–5, 2016.
[8] H. Strasdat, Sophus - Lie groups for 2D/3D Geometry, 2019. [Online]. Available: https://strasdat.github.io/Sophus/ (visited on 11/19/2019).
[9] S. Agarwal and K. Mierle, Ceres Solver — A Large Scale Non-linear Optimization Library, 2019. [Online]. Available: http://ceres-solver.org/ (visited on 11/19/2019).
[10] J. R. R. A. Martins, P. Sturdza, and J. J. Alonso, “The Complex-Step Derivative Approximation,” ACM Trans. on Mathematical Software, vol. 29, no. 3, pp. 245–262, 2003.
[11] W. Squire and G. Trapp, “Using Complex Variables to Estimate Derivatives of Real Functions,” SIAM Review, vol. 40, no. 1, pp. 110–112, 1998.
[12] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, “The EuRoC Micro Aerial Vehicle Datasets,” The International Journal of Robotics Research, vol. 35, no. 10, pp. 1157–1163, 2016.
[13] T. D. Barfoot, “AER 1513 Course Assignments,” in State Estimation for Aerospace Vehicles, Toronto, ON: Uni. of Toronto, 2009.
[14] G. S. Chirikjian, Stochastic Models, Information Theory, and Lie Groups - Volume 2. Birkhäuser Boston, 2009.
[15] B. Hall, Lie Groups, Lie Algebras and Representations, second. New York, NY: Springer, 2015.
[16] J. Arsenaunt, “Practical Considerations and Extensions of the Invariant Extended Kalman Filtering Framework,” M.A.Sc. Thesis, McGill University, 2019.
[17] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, “On-Manifold Preintegration for Real-Time Visual-Inertial Odometry,” IEEE Trans. Robotics, vol. 33, no. 1, pp. 1–21, 2017.
[18] MDO Laboratory, A Guide to the Complex-Step Derivative Approximation, 2017. [Online]. Available: http://mdolab.engin.umich.edu (visited on 10/31/2019).
[19] V. Vittaldev, R. P. Russell, N. Arora, and D. Gaylor, “Second-Order Kalman Filters Using Multi-Complex Step Derivatives,” Advances in the Astronautical Sciences, vol. 143, no. 1, pp. 1–16, 2012.
[20] V. Peretroukhin and J. Kelly, “DPC-Net: Deep Pose Correction for Visual Localization,” IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 2424–2431, 2018.
[21] K.-L. Lai and J. L. Crassidis, “Extensions of the First and Second Complex-Step Derivative Approximations,” J. Comp. Applied Math, vol. 219, no. 1, pp. 276–293, 2008.