2024 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)
July 15-19, 2024. Boston, MA, USA

Exp[licit]

An Educational Robot Modeling Software based on Exponential Maps

Johannes Lachner*, Moses C. Nah*, Stefano Stramigioli, Neville Hogan

***Abstract*— Deriving a robot's equations of motion typically requires placing multiple coordinate frames, commonly using the Denavit-Hartenberg convention to express the kinematic and dynamic relationships between segments. This paper presents an alternative using the differential geometric method of Exponential Maps, which reduces the number of coordinate frame choices to two. The traditional and differential geometric methods are compared, and the conceptual and practical differences are detailed. The open-source software, Exp[licit]^{TM}, based on the differential geometric method, is introduced. It is intended for use by researchers and engineers with basic knowledge of geometry and robotics and aims to serve as a supportive resource during the study of differential geometric approaches. Code snippets and an example application are provided to demonstrate the benefits of the differential geometric method and assist users to get started with the software.**

I. Introduction

In standard robotic textbooks, orthonormal coordinate frames are used to describe robot kinematics and dynamics [1], [2]. When the Denavit-Hartenberg (DH) convention is used, predetermined rules have to be followed to position the coordinate frames and express the translational and rotational relations between them.

While this approach remains widely prevalent in university-level robotics courses, it has several limitations. First, multiple conventions exist to define the coordinate frames. Within these conventions, different numbers of rules have to be applied. Some conventions need special treatment, e.g., for parallel axes where the description is not unique. Second, a large number of coordinate frames has to be placed. This becomes especially unwieldy for robots with many degrees of freedom (DOF). Third, the kinematics and dynamics are expressed with a fixed set of coordinate frames on the robot bodies. If the kinematics change, such as in re-configurable robots, a new set of DH-parameters must be assigned [3]. This requires additional efforts, including distinguishing between revolute and prismatic joints [4]. Fourth, the choices of task-related stationary and body-fixed frames are restricted which is disadvantageous for algorithms which describe the dynamics of multiple points on different robot bodies, e.g., for whole-body control [5].

*J. Lachner and M. C. Nah contributed equally

J. Lachner, M. C. Nah, and N. Hogan are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 USA

J. Lachner and N. Hogan are also with the Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139 USA

S. Stramigioli is with the Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, 7522 Enschede, The Netherlands

In contrast, Differential Geometry and Lie Group Theory can be used as a mathematical framework which lifts the coordinate-level descriptions to the more abstract space of manifolds [6]. Robot kinematics and dynamics can be described as actions on those manifolds [7]. This mathematical abstraction leads to a formulation that requires the least number of coordinate frames to represent the robot's kinematics and dynamics. The theoretical strengths of geometric methods have been shown in excellent textbooks [4], [8], [9] and tutorial papers [10]–[12]. Papers that compare traditional and geometric methods emphasize algorithmic and computational aspects [11], [13], [14] but detailed discussion of conceptual and practical differences (e.g., the brief overview in [4]) is rare.

Many powerful software tools exist for simulating and controlling robots [15]–[17]. Since these tools typically offer extensive features, they present an 'overhead cost' in learning how to use the software [18], [19]. This might impede first-time users, such as students who want to simulate a simple robot for a robotics class. For powerful software tools that use differential methods [19]–[22], the extensive features might make it harder for non-experts and beginners to compare those methods with traditional ones like DH.

The main contribution of this paper lies in bridging the gap between theoretical understanding and practical application, by comparing traditional and differential geometric methods in robotics. Section II introduces the background of the traditional and differential geometric methods, highlighting the conceptual differences in deriving robot kinematics and dynamics. The first part of Section III compares the two methods in terms of modularity, flexibility, and the number of coordinate frames required. The second part of Section III focuses on practical implementation. We introduce the software Exp[licit]^{TM}[1], a simple MATLAB robotic toolbox. It serves as a supportive resource during the study of differential geometric approaches in robotics. By stripping away the complexity often associated with more advanced differential geometric software, Exp[licit] should provide a straightforward, hands-on experience.

II. Derivation of Robot Kinematics and Dynamics

This section presents a detailed comparison of both approaches. The theoretical derivation is focused on the Forward Kinematic Map, Jacobian Matrix, and Mass Matrix of an open-chain n-DOF robot. A computational comparison with the RVC MATLAB toolbox [17] also includes the

[1]https://explicit-robotics.github.io/

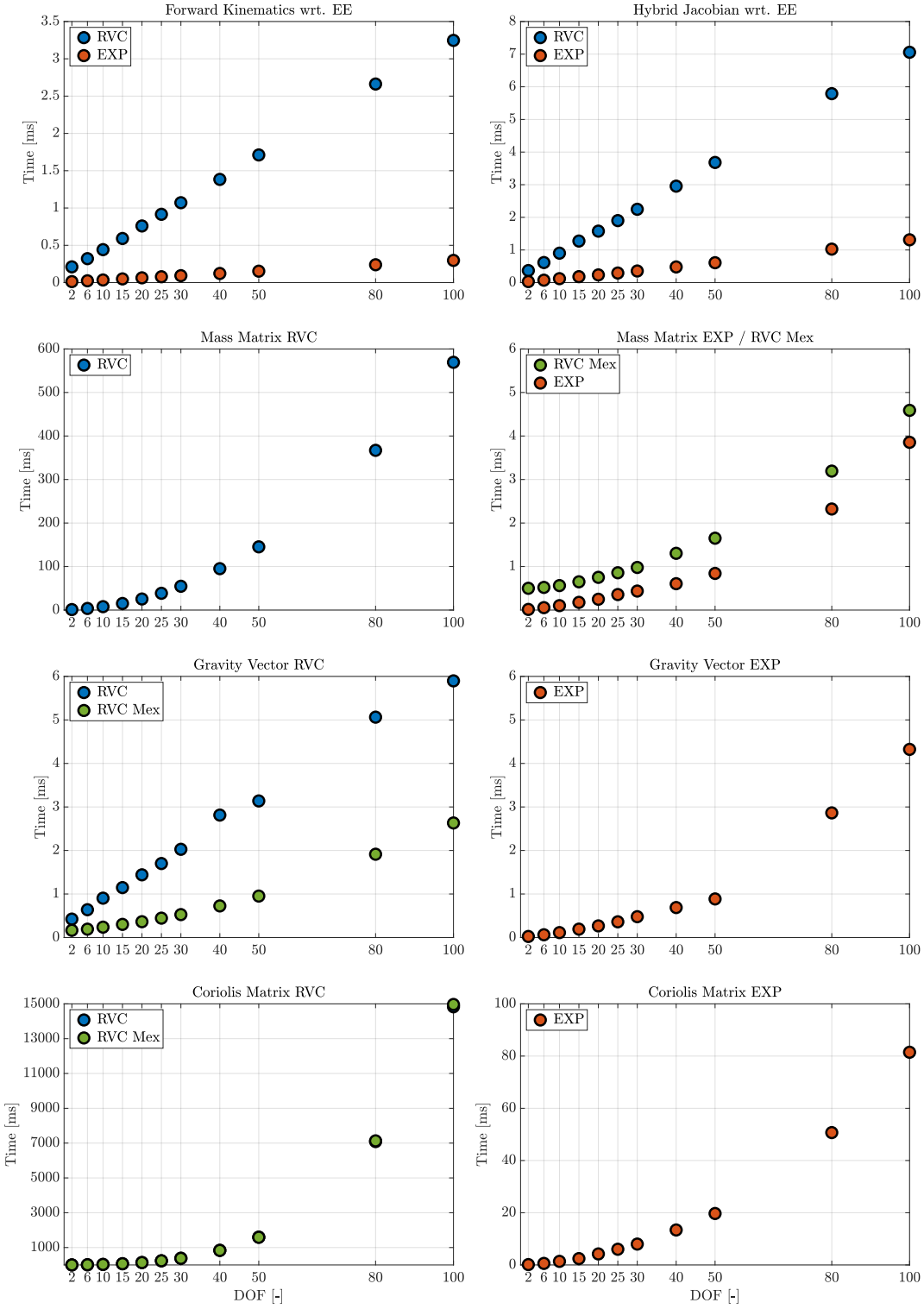

Fig. 1: Computation times of RVC and Exp[licit] for five kinematic and dynamic calculations. For the Mass Matrix, Gravity, and centrifugal/Coriolis terms, the `MEX`-file option of RVC was invoked. First row: Comparison for Forward kinematics map (left) and Hybrid Jacobian (right) with respect to end-effector by using native Matlab scripts; Second row: Comparison for Mass Matrix of RVC (left) against Mass Matrix of RVC-`MEX` and Exp[licit] (right); Third row: Comparison for Gravity vector for RVC and RVC-`MEX` (left) against Gravity vector of Exp[licit] (right); Fourth row: Comparison for Coriolis Matrix of RVC (left) against Coriolis Matrix of RVC-`MEX` and Exp[licit] (right). Note that RVC and RVC-`MEX` overlap in the Coriolis Matrix plot.

gravity and centrifugal/Coriolis terms (fig. 1). More details about the computational comparison are presented in sec. III-B.7.

A. Preliminaries

The set of all robot configurations $\boldsymbol{q}$ constitute the manifold $\mathcal{Q}$ and the set of all homogeneous transformations H constitute the manifold $SE(3)$. To represent the robot's workspace motion, either a stationary or body-fixed coordinate frame has to be chosen.[2] We assume one stationary frame $\{S\}$, attached to the fixed base of the robot. Moreover, we denote $\{B\}$ as a body-fixed frame, which can be attached to any point of the robot. Often, $\{B\}$ coincides with the tool center point (i.e., the end-effector) of the robot. In this case, we denote $\{B\}$ as $\{ee\}$.

For a given joint configuration $\boldsymbol{q} \in \mathcal{Q}$, the orientation and translation of $\{ee\}$ with respect to $\{S\}$ can be derived via the *Forward Kinematic Map*, $\mathcal{Q} \rightarrow SE(3)$ and represented by the *Homogeneous Transformation Matrix* ${}^{S}\boldsymbol{H}_{ee}(\boldsymbol{q}) = \begin{pmatrix} {}^{S}\boldsymbol{R}_{ee} & {}^{S}\boldsymbol{p}_{ee} \\ 0 & 1 \end{pmatrix} \in SE(3)$. Here, ${}^{S}\boldsymbol{R}_{ee} \in SO(3)$ is the *Rotation matrix* of $\{ee\}$ with respect to $\{S\}$ and ${}^{S}\boldsymbol{p}_{ee} \in \mathbb{R}^3$ is the translation from $\{S\}$ to $\{ee\}$.

For a given joint velocity $\dot{\boldsymbol{q}} \in \mathbb{R}^n$, the workspace velocity of the robot's end-effector can be derived via the *Hybrid Jacobian Matrix*[3] ${}^{H}\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$, and represented by a 6D-vector of workspace velocities, called *Spatial Velocity* ${}^{S}\boldsymbol{V}_{ee} = \begin{pmatrix} {}^{S}\boldsymbol{v}_{ee} \\ {}^{S}\boldsymbol{\omega} \end{pmatrix} \in \mathbb{R}^6$. Here, ${}^{S}\boldsymbol{V}_{ee}$ incorporates the linear velocity ${}^{S}\boldsymbol{v}_{ee} \in \mathbb{R}^3$ of the origin of $\{ee\}$ with respect to $\{S\}$ and the angular velocity ${}^{S}\boldsymbol{\omega} \in \mathbb{R}^3$ of the end-effector body, both expressed in $\{S\}$.

The total kinetic co-energy $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}$ of an n-DOF robot is the sum of all contributions of kinetic co-energy stored by individual bodies: $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$ [4]. The matrix $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is called the *Mass Matrix* of the robot.

B. Traditional Method

1) Forward Kinematic Map via DH-convention: The DH-convention [23] is widely used to derive the Forward Kinematic Map. It is a set of rules to place body-fixed frames on the robot, and to derive the parameters that describe the kinematic relation between adjacent frames [4]. Within the multiple DH-conventions [24], [25], we outline the modified DH-convention which consists of four DH-parameters: link length a, link twist α, link offset d, and joint angle θ [1], [4], [17].

To derive the DH-parameters, multiple frames have to be placed on each link using the following rules (fig. 2):

(i) Define frames $\{1\}, \{2\}, \cdots, \{n\}$ on each link, ordered from the base to the end-effector of the robot. Choose axis $\hat{Z}_i$ of frame $\{i\}$ to be aligned with the i-th joint. For a revolute (prismatic) joint, direction of $\hat{Z}_i$ is along the positive direction of rotation (translation).

[2]From now on, we use "frame(s)" to refer to "coordinate frame(s)".

[3]We elaborate the notion "Hybrid" in the next subsection. Moreover, superscript H denotes "Hybrid," rather than referring to a frame.

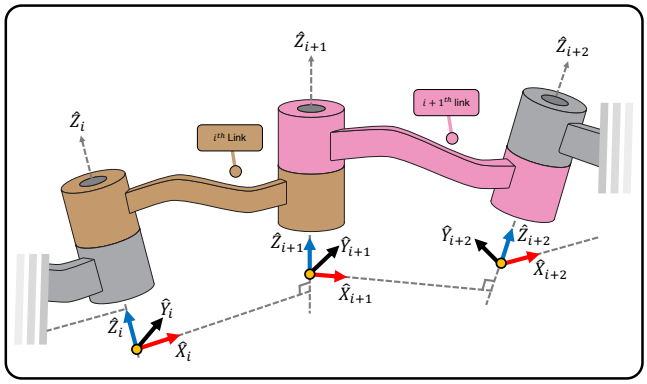

Fig. 2: Frames attached to an open-chain robot, using the DH-conventions.

(ii) For $i = 1, 2, ..., n-1$, find a line segment that is mutually perpendicular to axes $\hat{Z}_i$ and $\hat{Z}_{i+1}$. The intersection between this line and $\hat{Z}_i$ is the origin of frame $\{i\}$. Moreover, axis $\hat{X}_i$ is chosen to be aligned with this line segment, pointing from $\hat{Z}_i$ to $\hat{Z}_{i+1}$.

(iii) Attach the origin of frame $\{ee\}$ to the end-effector. To simplify the derivation of the DH-parameters, the $\hat{Z}_{ee}$ axis is usually chosen to be parallel to $\hat{Z}_n$ [1]. From $\hat{Z}_n$ and $\hat{Z}_{ee}$, $\hat{X}_n$ is defined using step (ii). Finally, choose $\hat{X}_{ee}$ such that valid DH-parameters can be defined [4].

(iv) The $\hat{Y}$ axes of frames $\{1\}, \{2\}, \cdots, \{n\}, \{ee\}$ are defined using the right-hand convention.

(v) Attach frame $\{S\}$ to the robot base. Usually, it is chosen to coincide with frame $\{1\}$ when joint 1 has zero displacement.

After assigning $n+2$ frames, $\{S\}, \{1\}, \cdots, \{n\}, \{ee\}$, the $4(n+1)$ DH-parameters can be expressed. With these parameters, the Homogeneous Transformation Matrix ${}^{i-1}\boldsymbol{H}_i \in SE(3)$ between frame $\{i-1\}$ and $\{i\}$ is defined for $i = 1, 2, ..., n+1$, where $\{0\} \equiv \{S\}$ and $\{n+1\} \equiv \{ee\}$. Finally, by concatenating these matrices, the Forward Kinematic Map, ${}^{S}\boldsymbol{H}_{ee}(\boldsymbol{q})$ can be derived:

$$ {}^{S}\boldsymbol{H}_{ee}(\boldsymbol{q}) = {}^{S}\boldsymbol{H}_1(q_1)\ {}^{1}\boldsymbol{H}_2(q_2)...{}^{n-1}\boldsymbol{H}_n(q_n)\ {}^{n}\boldsymbol{H}_{ee} \quad (1) $$

2) Jacobian Matrix by separating linear and angular velocities: To derive the Jacobian Matrix, the traditional method separately relates joint velocities to linear and angular workspace velocities [2]. We denote the linear and rotational part of the Jacobian as $\boldsymbol{J}(\boldsymbol{q})_v \in \mathbb{R}^{3\times n}$ and $\boldsymbol{J}(\boldsymbol{q})_\omega \in \mathbb{R}^{3\times n}$, respectively.

To derive $\boldsymbol{J}(\boldsymbol{q})_v$, the position ${}^{S}\boldsymbol{p}_{ee}$ has to be extracted from ${}^{S}\boldsymbol{H}_{ee}(\boldsymbol{q})$ (sec. II-B.1). Since ${}^{S}\boldsymbol{p}_{ee}$ is an analytical function of $\boldsymbol{q}$, $\boldsymbol{J}(\boldsymbol{q})_v$ collects the partial derivatives of ${}^{S}\boldsymbol{p}_{ee}$, with respect to the coordinate components of $\boldsymbol{q}$. Often, $\boldsymbol{J}(\boldsymbol{q})_v$ is called an "Analytical Jacobian" [2].

The matrix $\boldsymbol{J}(\boldsymbol{q})_\omega$ is commonly derived using a geometric method and specifying the frames based on DH-convention [2] (sec. II-B.1). More specifically, for $i = 1, 2, ..., n$:

- If the i-th joint is a revolute joint with unit-rotation axis ${}^{i}\hat{\boldsymbol{\omega}}_i$ expressed in $\{i\}$, the i-th column of $\boldsymbol{J}(\boldsymbol{q})_\omega$ is ${}^{S}\boldsymbol{R}_i\,{}^{i}\hat{\boldsymbol{\omega}}_i = {}^{S}\hat{\boldsymbol{\omega}}_i$.

- If the i-th joint is a prismatic joint, the i-th column of $\boldsymbol{J}(\boldsymbol{q})_\omega$ is a zero vector.

To calculate the spatial velocity ${}^S\boldsymbol{V}_{ee}$, $\boldsymbol{J}(\boldsymbol{q})_v$ and $\boldsymbol{J}(\boldsymbol{q})_\omega$ can be vertically concatenated:

$$ {}^S\boldsymbol{V}_{ee} = {}^H\boldsymbol{J}(\boldsymbol{q})\ \dot{\boldsymbol{q}} \tag{2} $$

Due to the analytical derivation of $\boldsymbol{J}(\boldsymbol{q})_v$ and the geometrical derivation of $\boldsymbol{J}(\boldsymbol{q})_\omega$, we call ${}^H\boldsymbol{J}(\boldsymbol{q})$ the *Hybrid Jacobian Matrix*.

3) Mass Matrix via Hybrid Jacobians: To derive the Mass Matrix of the robot, it is necessary to attach n additional frames to the center of mass (COM) of the n bodies. These will be denoted as $\{C_1\}$, $\{C_2\}$, $\cdots$, $\{C_n\}$, ordered from the base to the end-effector of the robot. The moment of inertia of the i-th body with respect to $\{C_i\}$ is denoted ${}^i\boldsymbol{\mathcal{I}}_i \in \mathbb{R}^{3\times3}$. To express ${}^i\boldsymbol{\mathcal{I}}_i$ in $\{S\}$, the rotation matrix ${}^S\boldsymbol{R}_i$ is used (sec. II-B.1): ${}^S\boldsymbol{\mathcal{I}}_i = {}^S\boldsymbol{R}_i\ {}^i\boldsymbol{\mathcal{I}}_i\ {}^S\boldsymbol{R}_i{}^T$.

For each body i, the Hybrid Jacobian Matrix ${}^H\boldsymbol{J}_i(\boldsymbol{q})$ is derived to describe the linear and angular velocity of $\{C_i\}$ with respect to $\{S\}$ (sec. II-B.2). Note that for each matrix ${}^H\boldsymbol{J}_i(\boldsymbol{q})$, the columns from $i+1$ to n are set to be zero since they do not contribute to the motion of body i [2].

Finally, for a given mass $m_i \in \mathbb{R}$ of the i-th body, $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ can be calculated by:

$$ \begin{aligned} \boldsymbol{M}(\boldsymbol{q}) = &\sum_{i=1}^{n} m_i\ \boldsymbol{J}_i(\boldsymbol{q})_v{}^T \boldsymbol{J}_i(\boldsymbol{q})_v \\ &+ \sum_{i=1}^{n} \boldsymbol{J}_i(\boldsymbol{q})_\omega{}^T\ {}^S\boldsymbol{\mathcal{I}}_i\ \boldsymbol{J}_i(\boldsymbol{q})_\omega \end{aligned} \tag{3} $$

C. Differential geometric method

1) Forward Kinematic Map via the Product of Exponentials Formula: For the geometric method, only two frames $\{S\}$ and $\{ee\}$ have to be chosen and assigned to the initial joint configuration of the robot $\boldsymbol{q}_0 \in \mathcal{Q}$. The initial Homogeneous Transformation Matrix is denoted ${}^S\boldsymbol{H}_{ee}(\boldsymbol{q}_0) \equiv {}^S\boldsymbol{H}_{ee,0} \in SE(3)$. In practice it is useful to select $\{S\}$ and $\{ee\}$ to have equal orientation (i.e., rotation matrix equals the identity matrix) such that only the translation between $\{S\}$ and $\{ee\}$ has to be identified to calculate ${}^S\boldsymbol{H}_{ee,0}$.

In the next step, the *Unit Joint Twists*[4] ${}^S\hat{\boldsymbol{\eta}}_i \in \mathbb{R}^6$ of each joint at initial joint configuration are expressed with respect to $\{S\}$. Depending on the type of the i-th robot joint, ${}^S\hat{\boldsymbol{\eta}}_i \in \mathbb{R}^6$ is defined by:

- If the i-th joint is a revolute joint, the unit-axis of rotation is ${}^S\hat{\boldsymbol{\omega}}_i$. *Any point* ${}^S\boldsymbol{p}_{\eta_i} \in \mathbb{R}^3$ along ${}^S\hat{\boldsymbol{\omega}}_i$ can be selected to define ${}^S\hat{\boldsymbol{\eta}}_i = (-[{}^S\hat{\boldsymbol{\omega}}_i]{}^S\boldsymbol{p}_{\eta_i},\ {}^S\hat{\boldsymbol{\omega}}_i)^T$. Here, $[{}^S\hat{\boldsymbol{\omega}}_i] \in so(3)$ is the skew-symmetric matrix form of ${}^S\hat{\boldsymbol{\omega}}_i$ [4]. The operation $[{}^S\hat{\boldsymbol{\omega}}_i]{}^S\boldsymbol{p}_{\eta_i}$ is equal to ${}^S\boldsymbol{\omega}\times{}^S\boldsymbol{p}_{\eta_i}$.
- If the i-th joint is a prismatic joint, the unit-axis of translation is ${}^S\hat{\boldsymbol{v}}_i$ and therefore ${}^S\hat{\boldsymbol{\eta}}_i = ({}^S\hat{\boldsymbol{v}}_i, \boldsymbol{0})$.

Note that the n Joint Twists ${}^S\hat{\boldsymbol{\eta}}_i$ are defined with respect to a single frame $\{S\}$. For most robots, the unit-axes of rotation (or translation) can be identified by visual inspection. The positions ${}^S\boldsymbol{p}_{\eta_i}$ can be determined by using CAD-programs.

Finally, the *Product of Exponentials Formula* [26] can be used to derive the Forward Kinematic Map:

$$ \begin{aligned} {}^S\boldsymbol{H}_{ee}(\boldsymbol{q}) = &\exp\left([{}^S\hat{\boldsymbol{\eta}}_1]q_1\right)\ \exp\left([{}^S\hat{\boldsymbol{\eta}}_2]q_2\right) \\ &\cdots \exp\left([{}^S\hat{\boldsymbol{\eta}}_n]q_n\right)\ {}^S\boldsymbol{H}_{ee,0} \end{aligned} \tag{4} $$

In this equation, $[{}^S\hat{\boldsymbol{\eta}}_i] \in se(3)$ is a 4×4 matrix representation of ${}^S\hat{\boldsymbol{\eta}}_i$ [8]. Given ${}^S\hat{\boldsymbol{\eta}} = ({}^S\boldsymbol{v}, {}^S\hat{\boldsymbol{\omega}})$ and $q \in \mathbb{R}$, a closed-form solution of $\exp([{}^S\hat{\boldsymbol{\eta}}]q)$ can be formulated [4]:

$$ \begin{aligned} &\exp([{}^S\hat{\boldsymbol{\omega}}]q) = \mathbb{I}_3 + \sin q[{}^S\hat{\boldsymbol{\omega}}] + (1-\cos q)[{}^S\hat{\boldsymbol{\omega}}]^2 \\ &\boldsymbol{G}(q) = \mathbb{I}_3 q + (1-\cos q)[{}^S\hat{\boldsymbol{\omega}}] + (q-\sin q)[{}^S\hat{\boldsymbol{\omega}}]^2 \\ &\exp\left([{}^S\hat{\boldsymbol{\eta}}]q\right) = \begin{bmatrix} \exp([{}^S\hat{\boldsymbol{\omega}}]q) & \boldsymbol{G}(q){}^S\boldsymbol{v} \\ \boldsymbol{0} & 1 \end{bmatrix} \end{aligned} \tag{5} $$

2) Jacobian Matrices via The Adjoint Map: For the geometric method, two Jacobian matrices exist: the Spatial Jacobian ${}^S\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$ and the Body Jacobian ${}^B\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$ [8]. The Spatial (respectively Body) Jacobian relates joint velocities $\dot{\boldsymbol{q}}$ to the Spatial (respectively Body) Twist ${}^S\boldsymbol{\xi}$ (${}^B\boldsymbol{\xi}$) [4], [8]:

$$ {}^S\boldsymbol{\xi} = \begin{bmatrix} {}^S\boldsymbol{v}_s \\ {}^S\boldsymbol{\omega} \end{bmatrix} = {}^S\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \qquad {}^B\boldsymbol{\xi} = \begin{bmatrix} {}^B\boldsymbol{v}_b \\ {}^B\boldsymbol{\omega} \end{bmatrix} = {}^B\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{6} $$

Here, ${}^S\boldsymbol{\omega}$ (respectively ${}^B\boldsymbol{\omega}$) is the angular velocity of the body, expressed in $\{S\}$ (respectively $\{B\}$); ${}^S\boldsymbol{v}_s$ is *not* the velocity of the origin of $\{S\}$, which is zero; it is the linear velocity of a point on the robot structure, viewed as if it travels through the origin of $\{S\}$ [4], [8]; ${}^B\boldsymbol{v}_b$ is the velocity of the origin of $\{B\}$ with respect to $\{S\}$, expressed in $\{B\}$ [4], [8].

The columns of ${}^S\boldsymbol{J}(\boldsymbol{q})$ and ${}^B\boldsymbol{J}(\boldsymbol{q})$ are derived using the Joint Twists $\hat{\boldsymbol{\eta}}_i$ and the *Adjoint Map* $\boldsymbol{Ad}_{\boldsymbol{H}} : \mathbb{R}^6 \to \mathbb{R}^6$ associated with $\boldsymbol{H} \in SE(3)$. As shown in [4], [8], [27], the Adjoint map is used to transform twists and wrenches between two frames. In matrix notation, $\boldsymbol{Ad}_{\boldsymbol{H}} = \begin{pmatrix} \boldsymbol{R} & [\boldsymbol{p}]\boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{pmatrix}$, with $[\boldsymbol{p}] \in \mathbb{R}^3$ being the skew-symmetric matrix form of position array and $\boldsymbol{R} \in SO(3)$ being the rotation matrix.

For planar robots, $\boldsymbol{\eta}_i'$ can be identified by visual inspection. In general, the i-th column $\boldsymbol{\eta}_i'$ of ${}^S\boldsymbol{J}(\boldsymbol{q})$ is:

$$ \boldsymbol{\eta}_i' = \begin{cases} {}^S\hat{\boldsymbol{\eta}}_1 & i = 1 \\ \boldsymbol{Ad}_{{}^S\boldsymbol{H}_{i-1}}\ {}^S\hat{\boldsymbol{\eta}}_i & i = 2, ..., n \end{cases} \tag{7} $$

In this equation, ${}^S\boldsymbol{H}_{i-1}$ can be derived via the Product of Exponentials Formula, i.e., ${}^S\boldsymbol{H}_{i-1} = \exp\left([{}^S\hat{\boldsymbol{\eta}}_1]q_1\right)\exp\left([{}^S\hat{\boldsymbol{\eta}}_2]q_2\right)\cdots\exp\left([{}^S\hat{\boldsymbol{\eta}}_{i-1}]q_{i-1}\right)$.

The i-th column $\boldsymbol{\eta}_i^\dagger$ of ${}^B\boldsymbol{J}(\boldsymbol{q})$ can be derived via $\boldsymbol{\eta}_i'$. With $\{B\}$ attached to the j-th body and $i \le j$:

$$ \boldsymbol{\eta}_i^\dagger = \boldsymbol{Ad}_{({}^i\boldsymbol{H}_j\ {}^S\boldsymbol{H}_{B,0})^{-1}}\ {}^S\hat{\boldsymbol{\eta}}_i \tag{8} $$

This shows the relation between ${}^S\boldsymbol{J}(\boldsymbol{q})$ and ${}^B\boldsymbol{J}(\boldsymbol{q})$. Here, ${}^i\boldsymbol{H}_j$ can be derived via the Product of Exponentials Formula (eq. (7)). Matrix ${}^S\boldsymbol{H}_{B,0} \in SE(3)$ is the Homogeneous Transformation of $\{B\}$ with respect to $\{S\}$ at initial joint configuration $\boldsymbol{q}_0$. For $j = 1, 2, ..., n-1$, the columns of ${}^B\boldsymbol{J}(\boldsymbol{q})$ from $j+1$ to n are zero.

[4]For simplicity, we will omit the term "Unit" in what follows.

3) Mass Matrix—Mapping Generalized Inertia with Body Jacobians: For the geometric method, the translational and rotational body contributions do not have to be separated. Instead, using the n frames $\{C_1\}$, $\{C_2\}$, ..., $\{C_n\}$ (sec. II-B.2), we define their corresponding Body Jacobian Matrices ${}^{B}\boldsymbol{J}_1(\boldsymbol{q})$, ${}^{B}\boldsymbol{J}_2(\boldsymbol{q})$, ..., ${}^{B}\boldsymbol{J}_n(\boldsymbol{q})$ (sec. II-C.2). Moreover, we use m_i and ${}^{i}\boldsymbol{\mathcal{I}}_i$ to define the *Generalized Inertia Matrix* $\boldsymbol{\mathcal{M}}_i = \begin{pmatrix} m_i\mathbb{I}_3 & \mathbf{0} \\ \mathbf{0} & {}^{i}\boldsymbol{\mathcal{I}}_{\boldsymbol{i}} \end{pmatrix} \in \mathbb{R}^{6\times 6}$ for each body i. In practice, $\{C_i\}$ are aligned with the principal moments of inertia. Hence, $\boldsymbol{\mathcal{M}}_i$ can be identified by using CAD-programs. Finally, the robot Mass Matrix can be calculated by:

$$\boldsymbol{M}(\boldsymbol{q}) = \sum_{i=1}^{n} {}^{B}\boldsymbol{J}_i(\boldsymbol{q})^T \; \boldsymbol{\mathcal{M}}_i \; {}^{B}\boldsymbol{J}_i(\boldsymbol{q}). \tag{9}$$

III. Exp[licit]: Concept, Features and Use-Cases

This section is split into two parts. First, we highlight the conceptual and practical differences between the traditional and geometric methods. To demonstrate the practical differences, we use a Franka robot.[5] Second, we introduce Exp[licit], a MATLAB-based robot software which leverages the advantages of the geometric method. By using Exp[licit], the model parameters of the Franka robot can be derived. The modular structure of Exp[licit] will be described by using code snippets and an example application. Finally, we compare the computational efficiency of Exp[licit] with the MATLAB-based open-source robotics software "Robotics, Vision and Control" (RVC) which is based on the DH-convention [17].

A. Conceptual and practical comparison between traditional and geometric methods

1) Forward Kinematic Map: The DH-convention provides a minimal parameter representation (four parameters) to define the Homogeneous Transformation Matrix [4]. This comes at a cost: a set of rules has to be carefully stipulated, which requires an extensive preparation in placing and transforming $n+2$ frames. If adjacent axes intersect or are parallel to each other, additional rules have to be considered to handle these exceptions for step (ii) in Section II-B.1 [1]. Since rotations and translations are only allowed along/about axes $\hat{X}$ and $\hat{Z}$, the choices for frames $\{S\}$ and $\{ee\}$ are restricted.

In contrast, the geometric method requires only two frames: the fixed inertial frame $\{S\}$ and the body-fixed frame $\{B\}$. Compared to the DH-approach, there are no restrictions on their position and orientation. The Product of Exponentials Formula provides considerable flexibility. To calculate the Joint Twists at initial configuration, any point on the twist axis can be chosen (sec. II-C.1). Once the Joint Twists are defined, the Forward Kinematic Map can be derived for any point on the robot structure (sec. III-B.6). This conceptual advantage yields a reduced computation time for the Forward Kinematic Map (sec. III-B.7)

[5] https://www.franka.de/

The practical benefit of the geometric method for the Franka robot can be seen in fig. 3. Compared to the DH-convention with nine frames [28], only two frames are needed. For our choice of initial configuration, the calculation of ${}^{S}\boldsymbol{H}_{ee,0}$ is straightforward since only the position of the end-effector has to be calculated. For our example, ${}^{S}\boldsymbol{p}_{ee,0} = (0.088, 0, 1.033)$ and ${}^{S}\boldsymbol{R}_{ee,0} = \mathbb{I}_3$.

The Joint Twists of the Franka robot are shown in table I. As can be seen, for a robot with revolute joints and an appropriate choice of initial configuration, the geometric approach requires at most four parameters (three translational parameters and one rotational parameter), similar to the DH approach. For prismatic joints, the geometric approach needs only three parameters.

Traditional Approach				
	a (m)	d (m)	α (rad)	θ (rad)
${}^{S}\boldsymbol{H}_1$	0.0	0.333	0	q_1
${}^{1}\boldsymbol{H}_2$	0.0	0.000	$-\pi/2$	q_2
${}^{2}\boldsymbol{H}_3$	0.0	0.316	$\pi/2$	q_3
${}^{3}\boldsymbol{H}_4$	0.0825	0.000	$\pi/2$	q_4
${}^{4}\boldsymbol{H}_5$	-0.0825	0.384	$-\pi/2$	q_5
${}^{5}\boldsymbol{H}_6$	0.0	0.000	$\pi/2$	q_6
${}^{6}\boldsymbol{H}_7$	0.088	0.000	$\pi/2$	q_7
${}^{7}\boldsymbol{H}_{ee}$	0.0	0.107	0	0
Geometric Approach				
${}^{S}\hat{\boldsymbol{\eta}}_1$	(0, 0, 0, 0, 0, 1)			
${}^{S}\hat{\boldsymbol{\eta}}_2$	(0.333, 0, 0, 0, -1, 0)			
${}^{S}\hat{\boldsymbol{\eta}}_3$	(0, 0, 0, 0, 0, 1)			
${}^{S}\hat{\boldsymbol{\eta}}_4$	(-0.649, 0, 0.0825, 0, 1, 0)			
${}^{S}\hat{\boldsymbol{\eta}}_5$	(0, 0, 0, 0, 0, 1)			
${}^{S}\hat{\boldsymbol{\eta}}_6$	(-1.033, 0, 0, 0, 1, 0)			
${}^{S}\hat{\boldsymbol{\eta}}_7$	(0, 0.088, 0, 0, 0, -1)			

TABLE I: Parameters for a Franka robot.

2) Jacobian Matrices: For the traditional method, the Hybrid Jacobian Matrix ${}^{H}\boldsymbol{J}(\boldsymbol{q})$ is separated into linear and angular parts. Before the linear part of ${}^{H}\boldsymbol{J}(\boldsymbol{q})$ can be derived, a choice for end-effector frame $\{ee\}$ has to be made. Although recalculating the velocities for a different frame on the end-effector body is straightforward, changing the frame to another point on any other robot body will require recalculating the position using the Forward Kinematic Map.

The geometric approach derived two different Jacobian matrices, ${}^{S}\boldsymbol{J}(\boldsymbol{q})$ and ${}^{B}\boldsymbol{J}(\boldsymbol{q})$. The basis of the derivation are the Joint Twists at initial configuration. Hence, no separation into linear and rotational parts is needed. ${}^{S}\boldsymbol{J}(\boldsymbol{q})$ and its output ${}^{S}\boldsymbol{\xi}$ (eq. (6)) only depend on one frame $\{S\}$. By using the Adjoint Map, ${}^{S}\boldsymbol{\xi}$ can be mapped to any point on the robot structure. By choosing a point equal to the origin of $\{ee\}$, the Spatial Velocity can be derived:

$${}^{S}\boldsymbol{V}_{ee} = \underbrace{\begin{pmatrix} \mathbb{I}_3 & -[{}^{S}\boldsymbol{p}_{ee}] \\ \mathbf{0} & \mathbb{I}_3 \end{pmatrix} {}^{S}\boldsymbol{J}(\boldsymbol{q})}_{{}^{H}\boldsymbol{J}(\boldsymbol{q})} \dot{\boldsymbol{q}}. \tag{10}$$

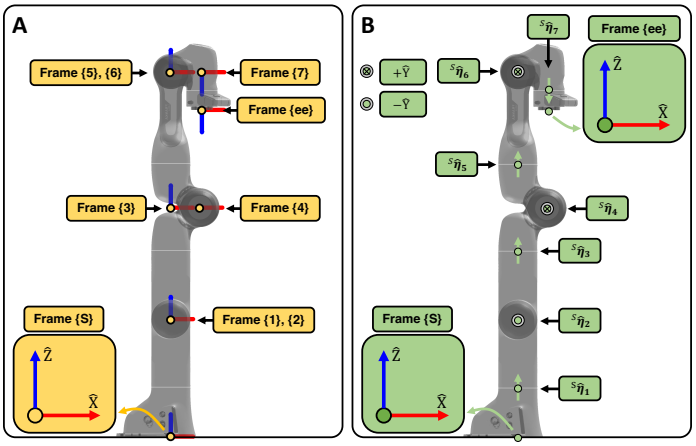

Fig. 3: Franka robot at initial configuration. The DH-convention is shown in (A) and the geometric method in (B). Only two frames are required for the geometric method (B). The frames shown in (A) are derived from [28].

Here, no modification of the Forward Kinematic Map is needed, which improves the length and clarity of the code and reduces the computation time of ${}^{H}\boldsymbol{J}(\boldsymbol{q})$ (sec. III-B.7).

3) Mass Matrix: For both approaches, the frames $\{C_i\}$ have to be attached to the COM of the robot at the initial configuration. For the traditional method, the orientation of these coordinate frames is restricted to obtain a valid set of DH-parameters. Commonly, $\{C_i\}$ is chosen to be aligned with frame $\{i\}$ (fig. 3A) and separately rotated by ${}^{S}\boldsymbol{\mathcal{I}} = {}^{S}\boldsymbol{R}_i\,{}^{i}\boldsymbol{\mathcal{I}}\,{}^{S}\boldsymbol{R}_i^{T}$.

For the geometric approach, the orientation of body frames $\{C_1\}$, $\{C_2\}$, ..., $\{C_n\}$ can be freely chosen. For each COM, the Body Jacobians are derived, again using the Adjoint Map (eqs. (7), (8)).

While the traditional method divides the derivation into linear and rotational contributions, the geometric method uses the generalized inertia matrices $\boldsymbol{\mathcal{M}}_i$ (eq. (9)) to derive the Mass Matrix. Even though $\boldsymbol{\mathcal{M}}_i$ may not be aligned with $\{S\}$, it need not be separately transformed. The transformation is incorporated in the map ${}^{B}\boldsymbol{J}_i(\boldsymbol{q})$.

B. Exp[licit]: Implementation of geometric method

The software can be installed from our Github repository: https://github.com/explicit-robotics/Explicit-MATLAB/. The documentation of the software can be found here: https://explicit-robotics.github.io/.

1) Software structure: The core of the software is the `RobotPrimitives`-class, which is used as the parent class of the software. It provides the member functions `getForwardKinematics`, `getSpatialJacobian`, `getHybridJacobian`, `getBodyJacobian`, `getMassMatrix`, `getGravityVector`, and `getCoriolisMatrix` for deriving the robot parameters. By inheriting the `RobotPrimitives`-class, a new robot class can be defined that shares the attributes and the member functions of the parent class. Each robot class brings its kinematic and dynamic properties (e.g., axes of rotation, link lengths, masses, etc.).

2) Initialization: Exp[licit] supports various 2D and 3D-robots (fig. 4). In this paper, we will use a Franka robot example (`franka.m`), which is inherited from the `RobotPrimitives`-class. The initialization is shown below:

```
% Call Franka Robot
robot = franka( );
robot.init( );
```

The `init`-function initializes all Joint Twists and Generalized Mass Matrices for the initial configuration (fig. 3).

3) Symbolic member functions: All member functions also accept symbolic arguments. This feature is helpful for control methods that require an analytical formulation of the robot's equations of motion, e.g., adaptive control methods [29]. An example to read out the symbolic form of the Forward Kinematics Map can be seen below:

```
% Create symbolic column vector
q_sym  = sym('q', [ robot.nq, 1 ]);

% Symbolic form of Hom. Trans. Matrix
H_ee_sym = robot.getForwardKinematics(
    q_sym );
```

4) Visualization and Animation: For visualization, the robot object can be passed to a 2D or 3D-animation object:

```
% Create animation
anim = Animation('Dimension', 3, 'xLim',
     [-0.7,0.7], 'yLim', [-0.7,0.7], '
    zLim', [0,1.4]);
anim.init( );
anim.attachRobot( robot )
```

The `Animation`-class heavily relies on MATLAB graphic functions (e.g., axes, patches, lighting). The key to our animation is to create a chain of transform objects (`hgtransforms`) instead of transforming vertices. The `Animation`-class has an optional input that allows the recording of videos with adjustable playback speeds.

At run-time (simulation time `t`), the robot object (in configuration `q`) and the animation can be updated:

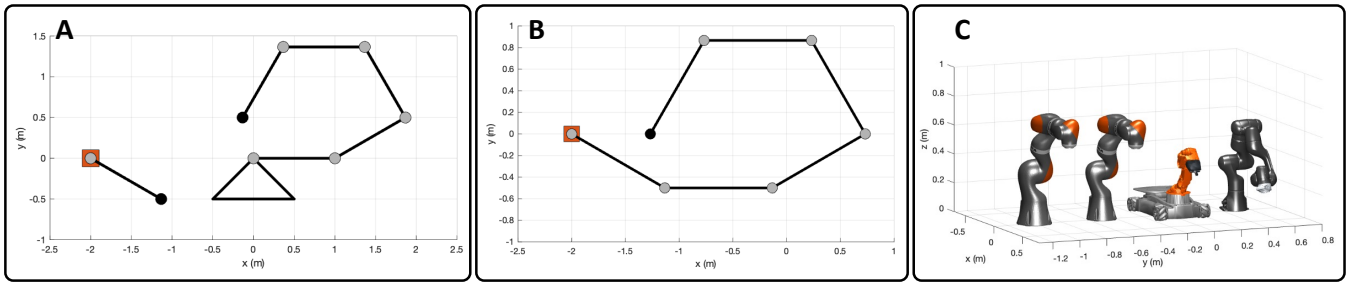

Fig. 4: Exp[licit] supports various 2D and 3D-robots. (A) Two planar robots: a Cart-Pole (left) and a Snake-Robot with variable DOF (right). (B) Two robots can be combined by using the `addKinematics`-method of the `RobotPrimitives`-class. In the example (B), the two robots of (A) are combined. (C) Currently supported 3D-robots: KUKA LBR iiwa (7 and 14 kg), YouBot, and Franka.

```
% Update kinematics
robot.updateKinematics(q);
anim.update(t);
```

5) Modularity through Joint Twists: The key to the modularity of Exp[licit] is the `setJointTwists( )`-function of the `RobotPrimitives`-class. So far, Exp[licit] supports revolute and prismatic joint types, indicated by the `JointTypes( )`-attribute. For each robot, the Joint Twists are derived from the joint directions (`AxisDirections`) and joint positions (`AxisOrigins`) in initial configuration. All member functions of the `RobotPrimitives`-class then re-use joints twists at runtime to map them from initial to current configuration (eq. (4) for Forward Kinematics, eq. (7) for Spatial Jacobian, and eq. (8) for Body Jacobian and Mass Matrix).

6) Example simulation: By default, the simulation loop is set to be real-time. It is beneficial to structure the simulation script the following way: (1) calculation of all kinematic and dynamic robot parameters; (2) trajectory generation; (3) control law; (4) integration and update. For (1), the member functions of the robot object can be used. Parts (2) and (3) are generally user specific. For the integration (4), any integrator can be used, e.g., MATLABs pre-built `ode45.m`.

To help users with parts (2) and (3), we implemented a simple impedance controller [30] for a Franka robot (`main_franka_IC.m`) that moves the end-effector around a circular path, while keeping its elbow position (joint four) fixed (fig. 5). Thanks to the modularity of the implemented geometric method, the kinematics of any point on any body can be selected by specifying the robot body (`'bodyID'`) and the corresponding position on the body (`'position'`):

```
% Get end-effector kinematics (default)
H_ee = robot.getForwardKinematics( q );
J_ee = robot.getHybridJacobian( q );

% Get kinematics of point on the elbow (
    body 4)
H_eb = robot.getForwardKinematics( q, '
    bodyID', 4, 'position', [-0.1,0,0] );
J_eb = robot.getHybridJacobian( q, '
    bodyID', 4, 'position', [-0.1,0,0] );
```

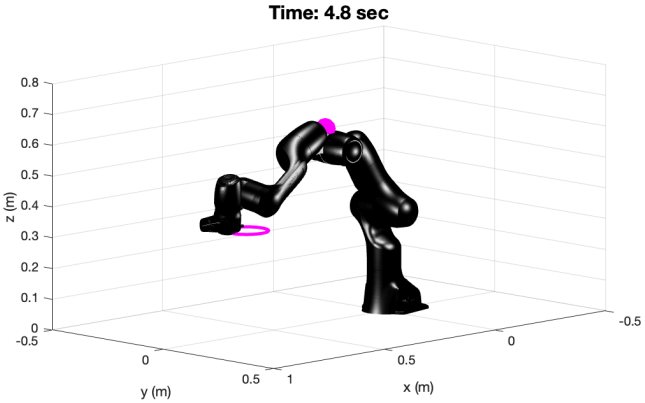

Fig. 5: Simulation of a simple impedance controller, using a Franka robot.

7) Comparison with MATLAB robotic toolbox: We compared the computational speed of Exp[licit] with the RVC MATLAB software [17], which uses the DH-convention. For RVC, version RTB10+MVTB4 (2017) was used.[6] By using native MATLAB scripts, the computation time was compared for the Forward Kinematic Map, Hybrid Jacobian, Mass Matrix, centrifugal/Coriolis terms, and Gravity vector of an n-DOF open-chain planar robot. The robot consisted of n identical uniform-mass bars with length $l = 1$m and mass $m = 1$kg. While Exp[licit] calculates the gravity and centrifugal/Coriolis terms with a closed-form algorithm, RVC uses recursive Newton-Euler methods (RNE). Both Exp[licit] and RVC used `.m`-MATLAB scripts. For the mass matrix, gravity and the centrifugal/Coriolis effects, the RVC-Method can invoke `MEX`-files to improve the computation speed. `MEX`-files are native `C` or `C++` files that are dynamically linked to the MATLAB application at runtime.

For the RVC software, the robot was constructed from the `SerialLink`-class which consists of n `Revolute`-classes. For Exp[licit], the robot was constructed from the `SnakeBot`-class (fig. 4A). Robots with various DOF were constructed and tested. The test was performed with a MacBook air (M1 Chip, 16GB Memory), using MATLAB 2022a. The `timeit()` function was used to measure the computation time.

[6] `https://petercorke.com/toolboxes/robotics-toolbox/`

The results of our computational comparisons are shown in fig. 1. For almost all computations, Exp[licit] was faster than the RVC software. Only for more than 70 DOF, was the gravity vector of the RVC `MEX`-file option faster than Exp[licit]. For both softwares, the computation of the Forward Kinematic Map and the Hybrid Jacobian showed a linear trend. The RVC software was capable of computing the Forward Kinematic Map of a 15-DOF robot within 1ms, whereas Exp[licit] required less than 0.5ms for more than 100 DOF. For the Hybrid Jacobian, the RVC software required more than 1ms for a 15-DOF robot, while Exp[licit] could accomplish the same for 80 DOF. The computation of the Mass Matrix showed an exponential trend for both softwares. While Exp[licit] outperformed RVC for MATLAB scripts by a factor of 100, RVC had a much better performance using `MEX`-files. Nevertheless, it was still slower than Exp[licit]. A similar trend was seen for the gravity vector: RVC's performance was improved by invoking `MEX`-files and showed better performance for more than 70 DOF. However, for the centrifugal/Coriolis terms, Exp[licit] drastically outperformed RVC.

These results highlight the computational advantages of a geometric approach, theoretically discussed in [31].

IV. Summary and Conclusion

This paper summarized and compared a traditional and a geometric method to derive the kinematic and dynamic parameters of an open-chain robot. We highlighted the conceptual and practical differences between the two approaches. While the geometric method demands a more abstract perspective (i.e., mapping of Joint Twists), we demonstrated several advantages compared to traditional methods. In summary, the advantages of the geometric method are: 1) Flexibility to express kinematic and dynamic relations without predefined rules and exceptions (sec. III-A); 2) Highly modular structure, since Joint Twists can be reused throughout the calculation (sec. III-B.5); 3) No more than two frames to describe robot kinematics and dynamics (fig. 3).

We introduced Exp[licit], a MATLAB-based toolbox that implements the geometric method and leverages its advantages. While acknowledging that our software does not encompass the extensive features of advanced robotic simulation tools [19]–[22], its strength lies in its simplicity and its focus on educational outcomes. It is tailored to foster an initial understanding and spark interest in the field of differential geometry for roboticists.

References

[1] J. J. Craig, *Introduction to robotics : mechanics & control / John J. Craig.* Reading, Mass.: Addison-Wesley Pub. Co.,, 1986.

[2] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[3] C. Nainer, M. Feder, and A. Giusti, "Automatic generation of kinematics and dynamics model descriptions for modular reconfigurable robot manipulators," *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 45–52, 2021.

[4] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

[5] C. Rocha, C. Tonetto, and A. Dias, "A comparison between the denavithartenberg and the screw-based methods used in kinematic modeling of robot manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 4, pp. 723–728, 2011.

[6] S. Stramigioli, "From differentiable manifolds to interactive robot control," Ph.D. dissertation, University of Delft, Netherlands, 1998.

[7] J. Lachner, "A geometric approach to robotic manipulation in physical human-robot interaction," Ph.D. dissertation, University of Twente, Netherlands, 2022.

[8] R. Murray, Z. Li, S. Sastry, and S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Taylor & Francis, 1994.

[9] J. M. Selig, *Geometric fundamentals of robotics*. Springer, 2005.

[10] S. Stramigioli and H. Bruyninckx, "Tutorial: Geometry and screw theory for robotics," in *2001 IEEE International Conference on Robotics and Automation (ICRA)*, 2001.

[11] A. Mueller, "Screw and lie group theory in multibody dynamics," *Multibody System Dynamics*, vol. 42, pp. 219–248, 2018.

[12] F. C. Park, B. Kim, C. Jang, and J. Hong, "Geometric Algorithms for Robot Dynamics: A Tutorial Review," *Applied Mechanics Reviews*, vol. 70, no. 1, p. 010803, 02 2018. [Online]. Available: https://doi.org/10.1115/1.4039078

[13] F. Park, "Computational aspects of the product-of-exponentials formula for robot kinematics," *IEEE transactions on automatic control.*, vol. 39, no. 3, pp. 643–647, 1994.

[14] A. Mueller, "Recursive second-order inverse dynamics for serial manipulators," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2483–2489.

[15] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[16] E. Rohmer, S. P. N. Singh, and M. Freese, "Coppeliasim: a versatile and scalable robot simulation framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[17] P. Corke and O. Khatib, *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*, ser. Springer Tracts in Advanced Robotics. Springer, 2011, vol. 73.

[18] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[19] M. L. Felis, "Rbdl: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, pp. 1–17, 2016.

[20] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer Publishing Company, Incorporated, 2016.

[21] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library : A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*, 2019, pp. 614–619.

[22] T. Koolen and R. Deits, "Julia for robotics: simulation and real-time control in a high-level programming language," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 604–611.

[23] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *Trans. ASME E, Journal of Applied Mechanics*, vol. 22, pp. 215–221, June 1955.

[24] J. Angeles, *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms (Mechanical Engineering Series)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[25] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[26] R. W. Brockett, *Robotic manipulators and the product of exponentials formula*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984.

[27] S. Stramigioli, *Modeling and IPC control of interactive mechanical systemsA coordinate-free approach*. Springer, 2001.

[28] F. Emika, "Robot and interface specifications," https://frankaemika.github.io/docs/control_parameters.html, 2027.

[29] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The international journal of robotics research*, vol. 6, no. 3, pp. 49–59, 1987.

[30] N. Hogan, "Impedance control (an approach to manipulation) part i, ii, iii," *Trans the ASME, J. of Dynamic systems, Measurement and Control*, vol. 107, pp. 1–24, 1985.

[31] F. C. Park, "Computational aspects of the product-of-exponentials formula for robot kinematics," *IEEE Transactions on Automatic Control*, vol. 39, no. 3, pp. 643–647, 1994.