Exponential Maps: An efficient, modular, and geometric method to calculate robot kinematics and dynamics

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

October 1, 2022

Part I: Theory

1 Introduction

A robot is a multi-domain system. The hardware of the robot (e.g., actuators and sensors) is part of the physical domain which interacts with the environment. The software of the robot controller is part of the information domain. To describe the physical interaction of the robot, the information domain has to represent the kinematic and dynamic attributes of the physical domain. This can be achieved by deriving a robot model that represents the physical robot properties by (co-)vectors and matrices. For robots with rigid bodies, the robot model can be described as a second order differential equations, called "the equations of motion":

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}^{\text{ext}}. \tag{1}$$

Here, $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ is the robot mass matrix, $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ is the centrifugal/Coriolis matrix, and $\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^{n}$ is the force resulting from gravitational potential energy. The control torque of the robot is denoted $\boldsymbol{\tau} \in \mathbb{R}^{n}$. For torque controlled robots, $\boldsymbol{\tau}$ determines the robot's workspace behavior. The workspace is a sub-space of the Special Euclidean Group $SE(3)$ (Section 3). The robot model can be used to transform the desired workspace behavior to the torque command. The co-vector $\boldsymbol{\tau}^{\text{ext}} \in \mathbb{R}^{n}$ incorporates the total resultant effects of all external forces as torques acting on the joints. These torques are not part of the robot model but can be acquired by sensors attached to the robot structure.

To be able to derive the robot model, many theoretical concepts have to be studied, e.g., linear algebra, geometry, and computer science. The better the theory is studied, the better[1] the roboticist is able to derive the components of Equation (1).

In standard robotic textbooks, one of the first concepts introduced are orthogonal coordinate frames with unit length (Siciliano et al., 2010). The translational and rotational relation between the coordinate frames is then used to describe the geometry of the robot kinematic structure. However, no deeper geometrical insights are given. This often leads to misunderstandings of basic geometric concepts. One popular misunderstandings is the notion of "position vectors," which is wrong in principle since positions do not constitute bases.

An efficient software for the calculation of rigid-body dynamics is provided by Felis (2016) which is based on the algorithms of Featherstone (2016). The computational efficiency was achieved by introducing "Spatial vectors," also called "6D-vectors."

In this article, we use differential geometry to show the geometric foundations of rigid-body dynamics. We show that the concept of 6D-vectors (Featherstone, 2016) can be derived from the well-established mathematical concepts of differential geometry and Lie algebra introduced in Levi (1905).

Differential geometry is an abstract concept that introduces actions on differentiable manifolds. This concept is powerful for robotics since the robot kinematics and dynamics are well described by actions on two distinct differentiable manifolds: the configuration manifold $\mathcal{Q}$ and the workspace manifold $W \subseteq SE(3)$. While the theoretical strengths of differential geometric methods are widely accepted in robotics, it is the intention of our two part tutorial paper to present their implementation advantages. Some implementation advantages were already discussed in Park (1994) and Mueller (2017, 2018, 2021).

[1]For instance, "better" can refer to the correctness of the model or to computational efficiency.

In part I of the paper, we present the differences between traditional methods of standard robotic textbooks and our geometric method. We highlight the current challenges in robotics using traditional methods (Section 2), and show that these challenges can be easily addressed using our geometric method (Section 4).

In Part II, we introduce our open-source robotics software **Exp[licit]™(EXP)** that is written based on Part I and makes use of **Exp***onential Maps*.[2] The computational benefits of our geometric method are presented in Section 5, where we compare our software with the well established open-source robotics software "Robotics, Vision and Control (RVC)" (Corke and Khatib, 2011). Both software are implemented in MATLAB (The MathWorks Inc., Natick, USA). The notation used in our software is mainly inspired by the great work of Murray et al. (1994). An overview about the nomenclature can be found in Appendix A.

Summarized, the contributions of this two-part tutorial are listed below:

1. Review and comparison of differential geometric methods with traditional methods to calculate robot kinematics and dynamics.
2. Development of MATLAB-based open-source software Exp[licit]™(EXP) based on differential geometric methods. Detailed documentation of EXP is provided to support non-expert users who want to create new robots.
3. Derivation of the model parameters for open-chain robots, especially useful for robots with non-equal joint parameters (i.e., revolute and prismatic joints).
4. Methods and code examples to add external kinematics to a given robot. This can be useful to add process-specific tools or merge multiple robot structures to one unified open-kinematic chain.
5. Methods and code examples to express the kinematics and dynamics with respect to any point on the robot structure. This can be useful for multi-task control, e.g., whole-body-control algorithms.
6. Comparison of computational efficiency between EXP and RVC.

2 Challenges in robotics from a traditional point of view

With the advance of machine learning and data science, the robotic field has extended rapidly. With more easy-to-use robots being available (El Zaatari et al., 2019), it is possible for non-experts to program a robot. Nevertheless, several unique challenges still make the analysis of robot kinematics and dynamics difficult.

Customize your robot

New manufacturing techniques make it possible for researches to design robots optimized for one specific application. During the prototype phase, the robot kinematics can be adapted quickly with rapid prototyping and 3D-printing. One of the challenges often encountered during this process is to exchange the type of the robot joints, e.g., changing one of the revolute joints to a prismatic joint. For example, through exchanging revolute and prismatic joints, the robot dexterity can be improved.

[2]The software can be downloaded via this link: LINK. A documentation of the code can be found here: LINK.

This is especially useful if the robot should reach in narrow spaces. One example is the research project “REFILLS” (EC-Project, 2017) which develops robots for supermarket logistics (Figure 1). Based on the task requirements, the robot kinematics are designed within a structure and dimension synthesis (Zumpe et al., 2019). In Section 4.1, we show that EXP is beneficial for such projects.

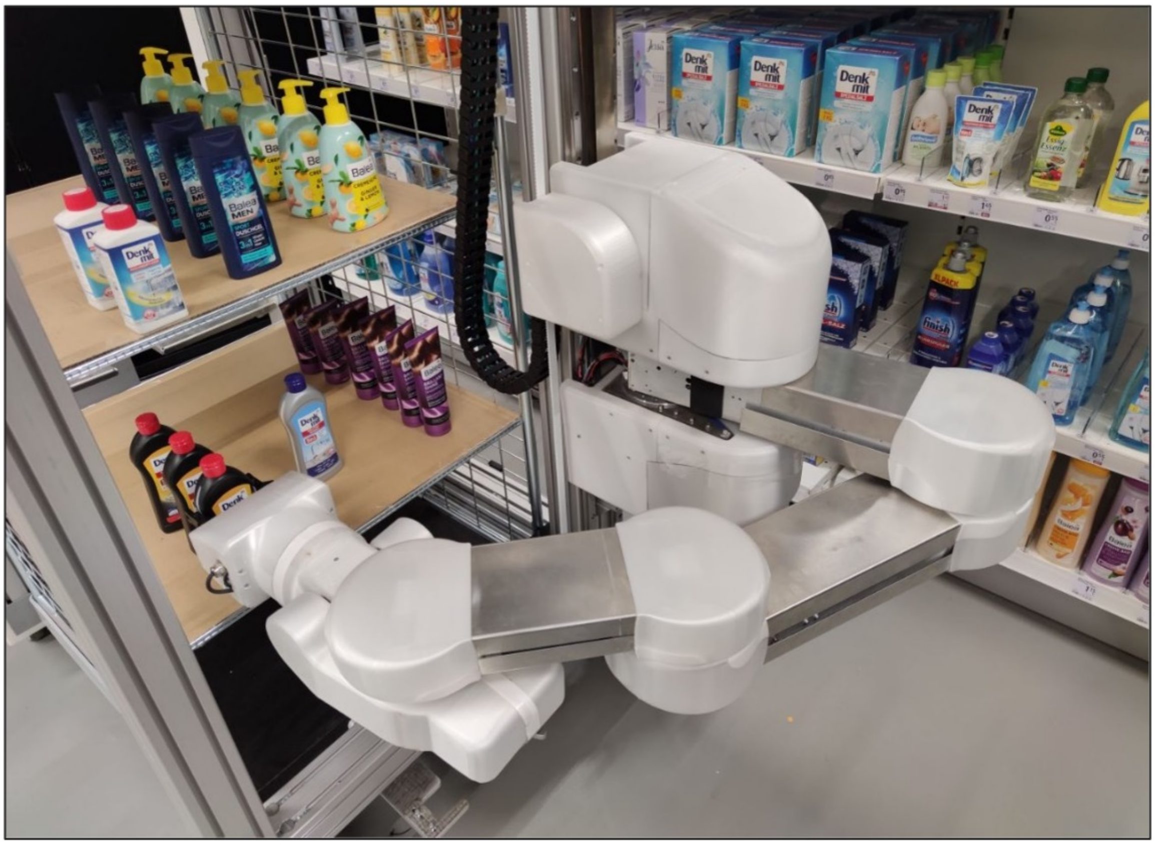

Figure 1: Prototype of a robot for supermarket logistics. The robot has two translational and four revolute joints (Zumpe et al., 2019).

Within EXP, revolute and prismatic joints can be exchanged by adapting one line of code.

Adding external kinematics

Traditional industrial robots are serial kinematic chains with six revolute joints. To enlarge the workspace, the robot can be equipped with external kinematics, e.g., linear axes, turntables, or mobile platforms. It is even possible to couple multiple robots (Figure 2, Section 4.2). For traditional robot controllers, it is hard to adapt the robot software in later development stages. This has the consequence that the robot and the external kinematics are controlled separately and are not treated as one holistic kinematic system. Hence, the robot’s dexterity cannot be used to a full extend.

A robot alone does not constitute a robot application. The robot has to be equipped with a process-specific tool. For special applications, actuated tools with multiple degrees of freedom (DOF) exist. An example for such an application is the research project “MURAB” (EC-Project,

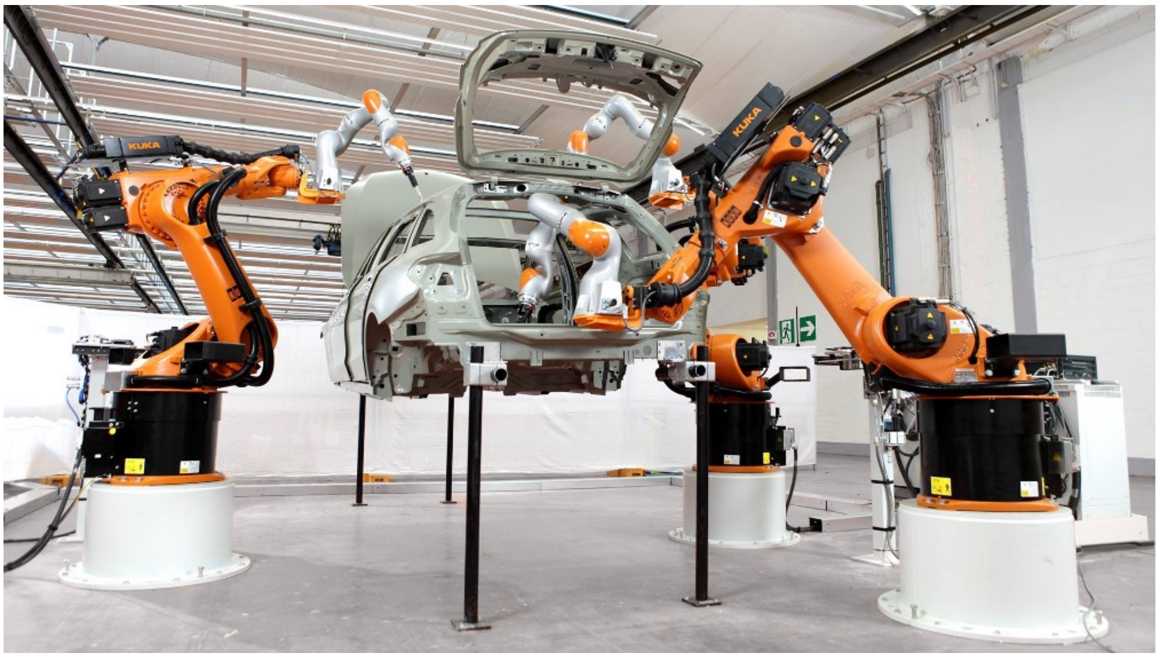

Figure 2: Laboratory experiment: Lightweight robots are mounted on the end-effector of KUKA KR60 robots. The application improves the corrosion protection of vehicle bodies (KUKA and Daimler, 2019).

2019) (Figure 3). A lightweight robot is equipped with a medical ultrasound probe to detect lesions in a female breast. Once a lesion has been detected, the three DOF of the tool can be used to position a needle guide for the surgeon. The surgeon will then perform a manual biopsy. For such applications, it is beneficial to include the tool kinematics in the kinematic chain of the robot. In Section 4.2, we show how EXP can be used to add external kinematics.

Controlling multiple points on the robot body

Traditionally, the robot software is focused on one process-specific task. The robot task is related to a body-fixed coordinate frame, e.g., placed on the tool tip called "tool center point." However, modern control algorithms make it possible to control multiple tasks (Lachner et al., 2022). These additional tasks are not necessarily related to the robot tool tip. For instance, a point on the robot elbow could be controlled to prevent collisions with an obstacle (Hjorth et al., 2021). With EXP, the kinematics and dynamics with respect to multiple points on the robot structure can be calculated. In Section 4.3, we show an example of how to control two robot tasks simultaneously.

3 Comparison of a traditional and a geometric method to calculate robot kinematics and dynamics

A robot motion can be represented by an action on a differentiable manifold. The robot's joint motion $\boldsymbol{q}(t)$ is a curve on the manifold $\mathcal{Q}$ and the robot's workspace (or end-effector) motion $\boldsymbol{H}(t)$

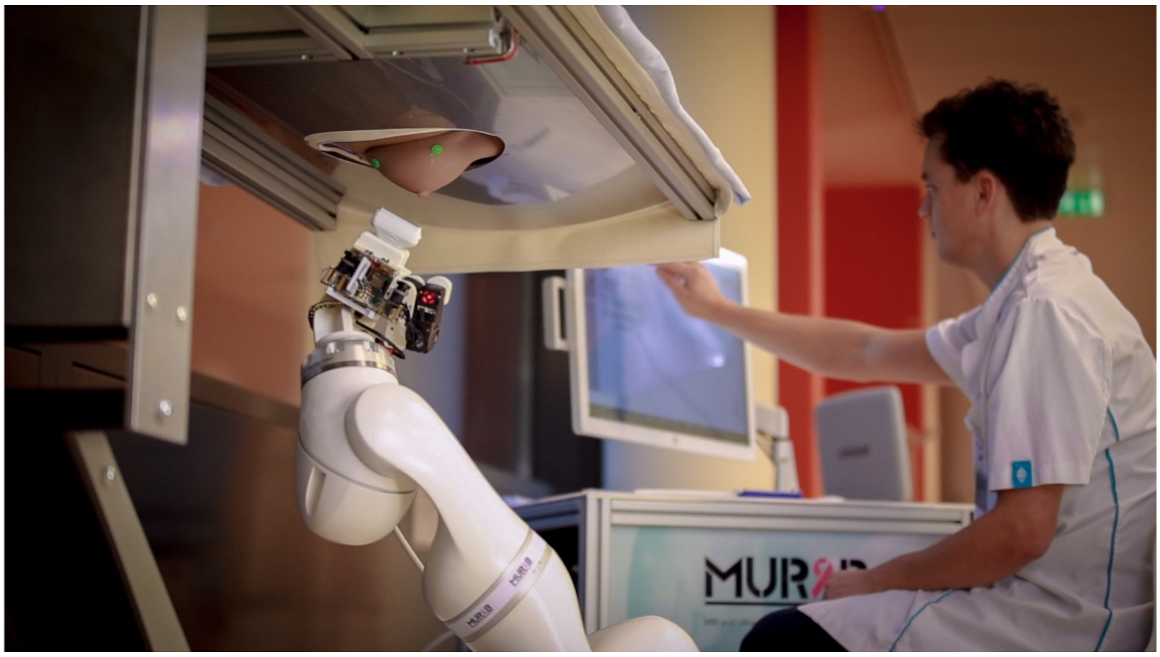

Figure 3: Prototype of a robot tool to detect lesions in a female breast. After ultrasound scanning, the surgeon can take a biopsy. A needle guide is provided by a tool with three DOF (Welleweerd, 2022).

is a curve on the manifold $SE(3)$ (Figure 4). Usually, the curve parameter used in robotics is time.

Roboticists are interested to represent an open set of the manifold in coordinate charts. Chart maps of an open set on $\mathcal{Q}$ yield joint coordinates and chart maps of an open set on $W \subseteq SE(3)$ yield workspace coordinates. Once a basis is chosen, vectors can be parameterized by coordinates. The joint velocity $\dot{\boldsymbol{q}}$ is a vector in $\mathbb{R}^n$. Vectors of $se(3) \cong \mathbb{R}^6$ can be represented with the help of *Cartesian coordinate frames.* A coordinate frame is a set of mutually orthogonal axes with unit length. To represent the velocity of the curve $\boldsymbol{H}(t)$, either a stationary or body-fixed coordinate frame has to be chosen with respect to which the velocity can be expressed. The former is called the *Spatial Twist* ${}^S\boldsymbol{\xi} \in se(3)$ (Figure 4) and the latter is called the *Body twist* ${}^B\boldsymbol{\xi} \in se(3)$. The motion of a robot joint can be represented by the *joint twist* $\boldsymbol{\eta} \in se(3)$ (Figure 4).

While coordinate frames are used excessively in the traditional robotic literature, differential geometric methods try to keep the coordinate dependency at a minimum. This is beneficial for the kinematic and dynamic calculation, as will be shown in the sequel of the paper.

This section will oppose a traditional method to a geometric method to receive the kinematic and dynamic parameters of a robot. The traditional method is based on well-established robotics literature, e.g., Angeles (2006); Siciliano and Khatib (2007); Siciliano et al. (2010) and Corke and Khatib (2011). The geometric method is based on Exponential Formula (Murray et al., 1994; Stramigioli and Bruyninckx, 2001). By showing the differences between both methods, the reader will gather the theoretical background to understand the implementation of the geometric method presented in Part II. We show that the reduced coordinate dependency of the geometric method has structural and computational advantages. For readers who want to study the differential geometric methods in more detail, we recommend the lecture of Stramigioli (2022) and the books of Murray

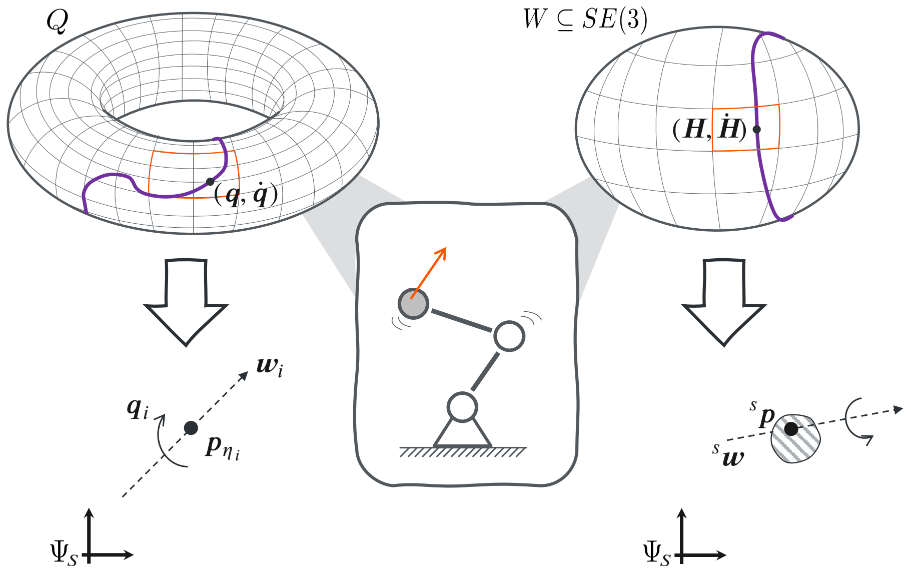

Figure 4: Coordinate chart of the configuration and workspace manifold (modified after Lachner (2022)). Once a stationary coordinate frame {S} is chosen, the joint motion can be represented by a joint twist (coordinate chart on the left) and the workspace motion can be represented by a Spatial twist (coordinate chart on the right).

et al. (1994) and Dubrovin et al. (2013).

3.1 Forward Kinematics Map

This sub-section shows two methods to compute the robot pose, expressed in a body-fixed coordinate frame on one of the robot bodies. For a single task, the body-fixed coordinate frame, {ee} usually coincides with the tool center point of the robot. For a given joint configuration $\boldsymbol{q} \in \mathcal{Q}$, the orientation and translation of the coordinate frame can be represented by the *Homogeneous transformation matrix* $^S\boldsymbol{H}_{ee} = \begin{pmatrix} ^{\boldsymbol{S}}\boldsymbol{R_{ee}} & \Delta\boldsymbol{p} \\ 0 & 1 \end{pmatrix} \in SE(3)$, which incorporates the *Rotation matrix* $^{\boldsymbol{S}}\boldsymbol{R_{ee}} \in SO(3)$ of {ee} with respect to the stationary coordinate frame {S} and the translation $\Delta\boldsymbol{p} \in \mathbb{R}^3$ between {ee} and {S}. The map from $\boldsymbol{q}$ to $^S\boldsymbol{H}_{ee}$ can be represented by the *Forward Kinematic Map* $L : \mathcal{Q} \to \mathbb{R}^3$.

Traditional method: Denavit-Hartenberg convention

The calculation of the Forward Kinematics Map traditionally involves the placement of a range of coordinate frames. Firstly, a stationary inertial coordinate frame {S} has to be chosen. Secondly, a body-fixed coordinate frame {ee} has to be placed on the end-effector body of the robot, e.g., placed on the robot's tool tip. Thirdly, body-fixed coordinate frames have to placed on (the center of) each joint axis. For each link i, the homogeneous transformation $^{i-1}\boldsymbol{H}_i \in SE(3)$ with respect to the previous link $i-1$ is calculated. Finally, these matrices are concatenated to express the homogeneous transformation of {ee} with respect to {S}:

$$^S\boldsymbol{H}_{ee} = \; ^S\boldsymbol{H}_1 \; ^1\boldsymbol{H}_2 \cdots \; ^{n-1}\boldsymbol{H}_{ee}, \tag{2}$$

where ${}^{S}\boldsymbol{H}_{ee} \in \; SE(3)$. To simplify the derivation of the Forward Kinematics Map, the Denavit-Hartenberg (DH) convention (Denavit and Hartenberg, 1955) can be used to parameterize the kinematic structure. The DH convention is a procedure which defines how to place the body-fixed coordinate frames on each joint axis such that the parameters to describe the transformation between two bodies can be reduced from six to four. Afterwards, the translations and orientations between two consecutive joints can be calculated. To express the Homogeneous transformation between two joints, four parameters have to be defined. Literature on DH convention can be found in Angeles (2006); Siciliano and Khatib (2007); Siciliano et al. (2010). A modified procedure for the DH convention was proposed by Craig (1986) and discussed in Corke and Khatib (2011).

Geometric method: Product of Exponentials Formula

For the geometric method, only two coordinate frames have to be chosen: {S} and {ee}. For a given initial joint configuration of the robot, the initial Homogeneous transformation can be expressed as ${}^{S}\boldsymbol{H}_{ee}(t_0) \in SE(3)$, where t_0 is the initial time parameter on the curve ${}^{S}\boldsymbol{H}_{ee}(t)$. In practice, it is convenient to select a outstretched robot configuration since in this way {S} and {ee} have equal orientation (identity rotation matrix) and only the translation between {S} and {ee} has to be considered to receive ${}^{S}\boldsymbol{H}_{ee}(t_0)$.

In the next step, the joint twists in initial joint configuration are calculated. A revolute joint rotates about an unit-axis $\hat{\boldsymbol{\omega}}_i$. A prismatic joint translates along the unit-axis $\hat{\boldsymbol{v}}_i$. Having selected a point $\boldsymbol{p}_{\eta_i} \in \mathbb{R}^3$ on the line along $\hat{\boldsymbol{\omega}}_i$, the joint twist $\boldsymbol{\eta}_i \in se(3)$ can be calculated. Here, $\boldsymbol{\eta}_i = (-\hat{\boldsymbol{\omega}}_i^{\sim}\boldsymbol{p}_{\eta_i} \;,\; \hat{\boldsymbol{\omega}}_i)^T$ for a revolute joint and $\boldsymbol{\eta}_i = (\hat{\boldsymbol{v}}_i \;,\; 0)^T$ for a prismatic joint, where $\hat{\boldsymbol{\omega}}_i^{\sim} \in so(3)$ is the skew-symmetric matrix form of $\hat{\boldsymbol{\omega}}_i$. Each $\boldsymbol{\eta}_i$ is expressed with respect to {S} solely.

When the joints move, the points (in Homogeneous coordinates) on each joint twist axis will be transformed from initial position $\boldsymbol{p}_{\eta_i}(t_0)$ to position $\boldsymbol{p}_{\eta_i}(t)$ after the motion. This can be represented by the Exponential Map $\boldsymbol{p}_{\eta_i}(t) = \exp(\boldsymbol{\eta}_i^{\sim} q_i)\; \boldsymbol{p}_{\eta_i}(t_0)$. Here, $\exp(\boldsymbol{\eta}_i^{\sim} q_i)$ is equal to the Homogeneous transformation $\boldsymbol{H}(\boldsymbol{\eta}_i, q_i) \in SE(3)$. The action of $\exp(\boldsymbol{\eta}_i^{\sim} q_i)$ is equal to a screw motion of the joint twist $\boldsymbol{\eta}_i$. The matrix $\boldsymbol{H}(\boldsymbol{\eta}_i, q_i)$ contains a rotational and translational part. The rotational part of $\boldsymbol{H}(\boldsymbol{\eta}_i, q_i)$ can also be represented by Exponential coordinates: $\boldsymbol{R}(\hat{\boldsymbol{\omega}}, q_i) = \exp(\hat{\boldsymbol{\omega}}_i^{\sim} q_i) \in SO(3)$. A revolute joint rotates q_i-radians about the unit-axis $\hat{\boldsymbol{\omega}}_i$. A closed-form solution for $\exp(\hat{\boldsymbol{\omega}}_i^{\sim} q_i)$ can be obtained via *Rodrigues' formula* (Murray et al., 1994). For prismatic joints, $\exp(\hat{\boldsymbol{\omega}}_i^{\sim} q_i)$ is equal to the identity. The translational part of $\boldsymbol{H}(\boldsymbol{\eta}_i, q_i)$ is simply $\hat{\boldsymbol{v}}_i q_i$ for prismatic joints (with unit [m]). For revolute joints, the translational part can be calculated by $-(\boldsymbol{I} - \exp(\hat{\boldsymbol{\omega}}_i^{\sim} q_i))\; (\hat{\boldsymbol{\omega}}_i^{\sim})^2 q_i$. Finally, the individual joint motions can be combined via the *Product of Exponentials Formula* (Brockett, 1984):

$$ {}^{S}\boldsymbol{H}_{ee}(t) = \exp(\boldsymbol{\eta}_1^{\sim} q_1)\; \exp(\boldsymbol{\eta}_2^{\sim} q_2) \cdots \exp(\boldsymbol{\eta}_n^{\sim} q_n)\; {}^{S}\boldsymbol{H}_{ee}(t_0). \tag{3} $$

Comparison of both methods

For the traditional method, the calculation of the Forward Kinematics Map needs extensive preparation. Firstly, coordinate frames have to be placed on each joint (n coordinate frames). Secondly, to calculate the Homogeneous transformation matrices between two consecutive joints via the DH convention, many (local) variables have to be introduced (for revolute joints: 4 times n parameters) (Mueller, 2018). Thirdly, these local variables differ for revolute and prismatic joints (Park, 1994).

The geometric method needs two coordinate frames. The homogeneous transformation matrix can be expressed as a function of the joint twist $\boldsymbol{\eta}_i$ and the joint motion q_i only. If an appropriate initial joint configuration ${}^{S}\boldsymbol{H}_{ee}(t_0)$ is chosen, the joint twist axes can be identified by visual inspection of the robot structure. "Appropriate" usually refers to a outstretched robot configuration (cf.

MATLAB code of EXP). All points on the joint twist axes in initial configuration can be extracted from CAD-models. Compared to the traditional method, far less variables have to be introduced for the geometric method. This is beneficial for the clarity of the code to calculate ${}^{S}\boldsymbol{H}_{ee}$ and makes the method computationally inexpensive (Section 5). The joint twists can be easily adapted in case revolute and prismatic joints should be exchanged (Section 4.1). This is especially useful during the prototype phase of the robot.

3.2 Workspace motion

This sub-section shows two methods to compute the robot's workspace motion for a given set of joint velocities $\dot{\boldsymbol{q}}$. The workspace motion can be represented by the *Spatial Velocity* ${}^{S}\boldsymbol{V}_{ee} = \begin{pmatrix} {}^{S}\boldsymbol{v}_{ee} \\ {}^{S}\boldsymbol{\omega} \end{pmatrix} \in \mathbb{R}^{6}$, which 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. The map from $\dot{\boldsymbol{q}}$ to ${}^{S}\boldsymbol{V}_{ee}$ can be represented by the *Jacobian matrix* $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{6\times n}$:

$$ {}^{S}\boldsymbol{V}_{ee} = \boldsymbol{J}(\boldsymbol{q})\, \dot{\boldsymbol{q}}. \tag{4} $$

Many different methods exist to calculate $\boldsymbol{J}(\boldsymbol{q})$ – some are analytical (Siciliano et al., 2010), some are geometrical (Angeles, 2006) and some are a mix of the latter (Siciliano et al., 2010). While the traditional method incorporates analytical and geometrical derivations, we show a purely geometric way to derive $\boldsymbol{J}(\boldsymbol{q})$.

Traditional method: Separation of linear and angular velocities

In calculus, a "Jacobian" is the partial derivative of a coordinate-valued function with respect to the coordinate components. This terminology was adapted in robotics and is used to describe the partial derivative of the Forward Kinematic Map (Section 3.1) $L : \mathcal{Q} \to \mathbb{R}^{3}$ with respect to the generalized joint coordinates $q_i \in \mathcal{Q}$ of the robot:

$$ \boldsymbol{J}(\boldsymbol{q})_{v} := \left(\frac{\partial L(\boldsymbol{q})}{\partial q_1}, \frac{\partial L(\boldsymbol{q})}{\partial q_2}, \dots, \frac{\partial L(\boldsymbol{q})}{\partial q_n} \right). \tag{5} $$

For a given set of joint velocities $\dot{\boldsymbol{q}}$, $\boldsymbol{J}(\boldsymbol{q})_{v} \in \mathbb{R}^{3\times n}$ can be used to calculate the linear velocity ${}^{S}\boldsymbol{v}_{ee} = \boldsymbol{J}(\boldsymbol{q})_{v}\, \dot{\boldsymbol{q}}$.

In many robotics papers, the definition of Equation (5) is used but the Jacobian is defined as an element of $\mathbb{R}^{6\times n}$ which maps $\dot{\boldsymbol{q}}$ to ${}^{S}\boldsymbol{V}_{ee}$. It is important to note that the Jacobian of Equation (5) can only be used to derive ${}^{S}\boldsymbol{v}_{ee}$. The angular velocity ${}^{S}\boldsymbol{\omega}$ cannot be calculated by the partial derivative of a function. An analytical function can only be derived for Euler angles. However, the partial derivative of this function will yield the time derivative of Euler angles which are unequal to ${}^{S}\boldsymbol{\omega}$. The map between $\dot{\boldsymbol{q}}$ to ${}^{S}\boldsymbol{\omega}$ will be denoted $\boldsymbol{J}(\boldsymbol{q})_{\omega} \in \mathbb{R}^{3\times n}$ in the further proceeding.

The derivation of the columns of $\boldsymbol{J}(\boldsymbol{q})_{\omega}$ starts by expressing the rotation matrix of joint i with respect to {S}: ${}^{S}\boldsymbol{R}_{i} = {}^{S}\boldsymbol{R}_{i-1}\, {}^{i-1}\boldsymbol{R}_{i}$. The time differentiation of ${}^{S}\boldsymbol{R}_{i}$ is: ${}^{S}\dot{\boldsymbol{R}}_{i} = \hat{\boldsymbol{r}}_{i}^{\sim}\, {}^{S}\boldsymbol{R}_{i}$, where $\hat{\boldsymbol{r}}_{i}^{\sim} \in \mathbb{R}^{3\times 3}$ is the matrix form of the unit-rotation-axis $\hat{\boldsymbol{r}} \in \mathbb{R}^{3}$ of joint i. Hence, the i-th column of $\boldsymbol{J}(\boldsymbol{q})_{\omega}$, denoted $\boldsymbol{\omega}_{i} \in \mathbb{R}^{3}$, can be derived by converting $\boldsymbol{\omega}_{i}^{\sim} = {}^{S}\boldsymbol{R}_{i}\, \hat{\boldsymbol{r}}_{i}^{\sim}\, \left({}^{S}\boldsymbol{R}_{i}\right)^{T}$ to vector form. For prismatic joints, the columns of $\boldsymbol{J}(\boldsymbol{q})_{\omega}$ are equal to a vector of zeros. Note that the calculation of $\boldsymbol{\omega}_{i}$ can be simplified using the DH-convention since $\hat{\boldsymbol{r}}_{i}$ is equal to the third column of the matrix ${}^{S}\boldsymbol{R}_{i}$ (Siciliano et al., 2010; Corke and Khatib, 2011).

The algebraic relation of Equation (5) can be solved by looking at the linear velocity contribution of each joint. For a revolute joint, the i-th columns of $\boldsymbol{J}(\boldsymbol{q})_{v}$ can be derived by the cross product of the unit-rotation axis $\hat{\boldsymbol{\omega}}_{i-1} \in \mathbb{R}^3$ of joint $i-1$ and the position difference between {ee} and the body-fixed coordinate frame placed on (the center of) joint $i-1$: $\hat{\boldsymbol{\omega}}_{i-1} \times ({}^S\boldsymbol{p}_{ee} - {}^S\boldsymbol{p}_{i-1})$. For a prismatic joint, the i-th columns of $\boldsymbol{J}(\boldsymbol{q})_v$ is simply $\hat{\boldsymbol{v}}_{i-1} \in \mathbb{R}^3$, the unit-axis of translation.

Finally, the matrices $\boldsymbol{J}(\boldsymbol{q})_v$ and $\boldsymbol{J}(\boldsymbol{q})_\omega$ are concatenated. Due to the analytical relation of $\boldsymbol{J}(\boldsymbol{q})_v$ and the geometrical derivation of $\boldsymbol{J}(\boldsymbol{q})_\omega$, we will call this matrix *Hybrid Jacobian Matrix* ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q}) = \begin{pmatrix} \boldsymbol{J}(\boldsymbol{q})_v \\ \boldsymbol{J}(\boldsymbol{q})_\omega \end{pmatrix}$.

Geometric method: Adjoint mapping of joint twists

In Section 3.1, the joint twists $\boldsymbol{\eta}_i$ were used to derive $\boldsymbol{H}(t)$.[3] We can map $\boldsymbol{\eta}_i$ from the initial robot configuration to the instantaneous robot configuration. This yields the joint twist $\boldsymbol{\eta}'_i \in se(3)$, which can be derived via *Adjoint mapping* $\boldsymbol{Ad}_H : se(3) \to se(3)$. Here, $\boldsymbol{Ad}_H$ can be represented in matrix notation as $\boldsymbol{Ad}_H = \begin{pmatrix} \boldsymbol{R} & \tilde{\boldsymbol{p}}\boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{R} \end{pmatrix}$, where $\boldsymbol{R} \in SO(3)$ and $\boldsymbol{p} \in \mathbb{R}^3$ are extracted from $\boldsymbol{H}(t)$. The homogeneous transformation of joint $i-1$ with respect to joint 1 can be expressed as the product: $\exp(\boldsymbol{\eta}_1^{\sim} q_1) \cdots \exp(\boldsymbol{\eta}_{i-1}^{\sim} q_{i-1})$. This action on $SE(3)$ is the input for $\boldsymbol{Ad}_H$ and therefore the instantaneous joint twist can be derived by $\boldsymbol{\eta}'_i = \boldsymbol{Ad}_{\exp(\boldsymbol{\eta}_1^{\sim} q_1)\cdots\exp(\boldsymbol{\eta}_{i-1}^{\sim} q_{i-1})}\, \boldsymbol{\eta}_i$. The joint twist $\boldsymbol{\eta}'_i$ has an important property: It is the i-th column of the *Spatial Jacobian Matrix* ${}^{\mathrm{S}}\boldsymbol{J}(\boldsymbol{q}) = (\boldsymbol{\eta}_1\ \boldsymbol{\eta}'_2 \cdots \boldsymbol{\eta}'_n)$. The map ${}^{\mathrm{S}}\boldsymbol{J}(\boldsymbol{q})$ can be used to transform joint velocities $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ to Spatial Twists ${}^{\mathrm{S}}\boldsymbol{\xi} \in se(3)$:

$$ {}^{\mathrm{S}}\boldsymbol{\xi} = {}^{\mathrm{S}}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}. \tag{6} $$

Finally, the Spatial velocity ${}^S\boldsymbol{V}_{ee}$ can be derived by:

$$ {}^S\boldsymbol{V}_{ee} = \underbrace{\begin{pmatrix} \boldsymbol{I} & -{}^S\boldsymbol{p}_{ee}^{\sim} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}}_{\boldsymbol{A}} {}^{\mathrm{S}}\boldsymbol{\xi} = \underbrace{\begin{pmatrix} \boldsymbol{I} & -{}^S\boldsymbol{p}_{ee}^{\sim} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} {}^{\mathrm{S}}\boldsymbol{J}(\boldsymbol{q})}_{{}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})}\ \dot{\boldsymbol{q}}. \tag{7} $$

Note that no rotation is involved in the mapping $\boldsymbol{A}$ of Equation (7) since both, the linear velocity part of $\boldsymbol{V}$ and the linear part of ${}^{\mathrm{S}}\boldsymbol{\xi}$ are expressed with respect to {S}. However, $\boldsymbol{V}$ and ${}^{\mathrm{S}}\boldsymbol{\xi}$ have different meanings: The former is the linear velocity of the origin of {ee} with respect to {S} and the latter is the instantaneous velocity of a point on the end-effector body as viewed travelling through {S} (Murray et al., 1994). It can be seen in Equation (7) that the Hybrid Jacobian Matrix can be derived by ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q}) = \boldsymbol{A}\ {}^{\mathrm{S}}\boldsymbol{J}(\boldsymbol{q})$.

Comparison of both methods

For the linear part of ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$, the traditional method needs a choice for many body-fixed coordinate frames and positions on the respective joint axes. Moreover, an end-effector position has to be fixed at the beginning of the calculation of ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$. If the point on the body fixed coordinate frame changes, a new coordinate frame has to be chosen and the DH-parameters have to be adapted. To calculate ${}^S\boldsymbol{V}_{ee}$, ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$ has to be modified by: ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})' = \begin{pmatrix} \boldsymbol{R}' & 0 \\ 0 & \boldsymbol{R}' \end{pmatrix} {}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$.

The geometric method maps joint twists from initial robot configuration to instantaneous robot configuration. Only one Spatial coordinate frame has to be chosen. This is a subtle but important difference to the traditional method, since ${}^S\boldsymbol{V}_{ee}$ can be calculated for any point on any robot body. Compared to the traditional method, the geometric method doesn't need a separation into linear

[3]For simplicity, the notion of coordinate frames is left out of this sub-sections.

and rotational part. This is especially beneficial for the length and clarity of the code to calculate ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$ (Section 4) and reduces the calculation time in case of robots with many DOF (Section 5).

3.3 Robot mass matrix

This sub-section shows a traditional and geometric method to compute the robot mass matrix (Equation (1)). The total kinetic energy of the robot is the sum of all contributions of kinetic energy performed by the individual bodies. This makes it possible to separate the computation of the mass matrix for each individual body. The robot's mass matrix is then equal to the sum of the mass matrices of all bodies, expressed with respect to {S}. For computation, we place a body-fixed coordinate frame $\{B_i\}$ on each robot body such that it coincides with the central axes of the body inertia. The mass of body i is represented by $m_i \in \mathbb{R}$. The inertia matrix with respect to the center of mass of each body is a diagonal matrix $\boldsymbol{\mathcal{I}}_i$ with the inertia values $I_{xi}, I_{y_i}, I_{zi} \in \mathbb{R}$ on the main diagonal.

Traditional method: Separation of masses and inertia

For the traditional method, the calculation of the mass matrix is separated for translational and rotational contributions of each body. The linear velocity of the origin of the center of mass of the body i is ${}^{S}\boldsymbol{v}_i = {}^{H}\boldsymbol{J}(\boldsymbol{q})_{v_i}\ \dot{\boldsymbol{q}}$. The angular velocity of body i is ${}^{S}\boldsymbol{\omega}_i = {}^{H}\boldsymbol{J}(\boldsymbol{q})_{\omega_i}\ \dot{\boldsymbol{q}}$. The matrices ${}^{H}\boldsymbol{J}(\boldsymbol{q})_{v_i}$ and ${}^{H}\boldsymbol{J}(\boldsymbol{q})_{\omega_i}$ are calculated separately, as shown in the traditional method of Section 3.2. However, the Jacobian matrices used to compute the mass matrix have to be adapted since the columns of ${}^{H}\boldsymbol{J}(\boldsymbol{q})_{v_i} \in \mathbb{R}^3$ and ${}^{H}\boldsymbol{J}(\boldsymbol{q})_{\omega_i} \in \mathbb{R}^3$ are arrays of zeros for all columns greater than i. This makes sense since only the first i moving bodies contribute to the velocities ${}^{S}\boldsymbol{v}_i$ and ${}^{S}\boldsymbol{\omega}_i$. The inertia matrix with respect to {S} is ${}^{S}\boldsymbol{\mathcal{I}} = {}^{S}\boldsymbol{R}_i\ \boldsymbol{\mathcal{I}}_i\ {}^{S}\boldsymbol{R}_i{}^T$. The rotation matrix ${}^{S}\boldsymbol{R}_i$ can again be calculated by using the DH-convention (Section 3.2). The kinetic energy of body i is: $T_i = \frac{1}{2}\dot{\boldsymbol{q}}^T\ \boldsymbol{J}(\boldsymbol{q})_{v_i}{}^T\ m_i\ \boldsymbol{J}(\boldsymbol{q})_{v_i}\ \dot{\boldsymbol{q}} + \frac{1}{2}\dot{\boldsymbol{q}}^T\ \boldsymbol{J}(\boldsymbol{q})_{\omega_i}{}^T\ {}^{S}\boldsymbol{\mathcal{I}}\ \boldsymbol{J}(\boldsymbol{q})_{\omega_i}\ \dot{\boldsymbol{q}}$.[4] Comparing with the well known equation for kinetic energy $T(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\dot{\boldsymbol{q}}^T\ \boldsymbol{M}(\boldsymbol{q})\ \dot{\boldsymbol{q}}$, it can be concluded that the robot mass matrix $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$ can be derived by summing the individual body contributions:

$$\boldsymbol{M}(\boldsymbol{q}) = \sum_{i=1}^{n} \boldsymbol{J}(\boldsymbol{q})_{v_i}{}^T\ m_i\ \boldsymbol{J}(\boldsymbol{q})_{v_i} + \boldsymbol{J}(\boldsymbol{q})_{\omega_i}{}^T\ {}^{S}\boldsymbol{\mathcal{I}}\ \boldsymbol{J}(\boldsymbol{q})_{\omega_i}. \tag{8}$$

Geometric method: Mapping Generalized Inertia with Body Jacobians

For the geometric method, the translational and rotational body contributions do not have to be separated. Instead, we will concatenate m_i and $\boldsymbol{\mathcal{I}}_i$ to get the *Generalized Inertia Matrix* $\boldsymbol{\mathcal{M}}_i \in \mathbb{R}^{6\times 6}$ for each body:

$$\boldsymbol{\mathcal{M}}_i = \begin{pmatrix} \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{pmatrix} & \boldsymbol{0} \\ \boldsymbol{0} & \begin{pmatrix} I_{xi} & 0 & 0 \\ 0 & I_{y_i} & 0 \\ 0 & 0 & I_{zi} \end{pmatrix} \end{pmatrix}.$$

For the Spatial Jacobian matrix in Section 3.2, we calculated the instantaneous joint twists $\boldsymbol{\eta}_i'$ which are expressed with respect to {S}. In this sub-section, $\boldsymbol{\eta}_i$ will be expressed with respect to

[4]For simplicity, the superscript "H" of the Jacobian matrices was kept out of the equation.

{B}: $\boldsymbol{\eta}_i^\dagger = \boldsymbol{Ad}^{-1}_{({}^S\boldsymbol{H}_B)}\ \boldsymbol{\eta}_i$. For the input of the inverse Adjoint mapping $\boldsymbol{Ad}^{-1}_{({}^S\boldsymbol{H}_B)} : se(3) \to se(3)$, Exponential Formula can be used to calculate the Homogeneous transformation matrix ${}^S\boldsymbol{H}_B(t) = \exp(\boldsymbol{\eta}_1^\sim q_1) \cdots \exp(\boldsymbol{\eta}_i^\sim q_i)\ {}^S\boldsymbol{H}_B(t_0)$. Here, ${}^S\boldsymbol{H}_B(t_0)$ is the Homogeneous transformation of {B} in the initial robot configuration (initial time parameter t_0). The joint twists $\boldsymbol{\eta}_i^\dagger$ have an important property: They are the columns of the *Body Jacobian matrix* ${}^B\boldsymbol{J}(\boldsymbol{q})_i \in \mathbb{R}^{6\times n}$ for body i: ${}^B\boldsymbol{J}(\boldsymbol{q})_i = (\boldsymbol{\eta}_1^\dagger, ..., \boldsymbol{\eta}_i^\dagger, \boldsymbol{0}, ..., \boldsymbol{0})$. Note that all columns greater than i are equal to a 6D-array of zeros. With ${}^B\boldsymbol{J}(\boldsymbol{q})_i$, the robot mass matrix can be calculated by

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

Comparison of both methods

For both methods, the contribution of the individual robot bodies can be analyzed. The traditional method separates the translational and rotational contributions of the bodies. Moreover, these contributions are separated for masses and inertia. This has calculational disadvantages, as will be shown in Section 5. The geometric method re-uses the joint twists and Exponential coordinates to express the Forward Kinematic Map and the Spatial Jacobian. This is especially beneficial for the length of the code and for the calculation time of $\boldsymbol{M}(\boldsymbol{q})$ (Section 5).

3.4 Coriolis and centrifugal matrix

Multiple ways exist to derive Coriolis and centrifugal terms of the robot. Moreover, the Coriolis and centrifugal terms can be represented as a matrix or a co-vector. The matrix $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$ is dependent on the inertial properties of the manipulator. It can be derived by the partial derivatives of the mass matrix $\boldsymbol{M}(\boldsymbol{q})$, with respect to $\boldsymbol{q}$ (Siciliano et al., 2010; Murray et al., 1994):

$$c_{ij}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\sum_{k=1}^{n}\left(\frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} + \frac{\partial M_{kj}}{\partial q_i}\right)\dot{q}_k, \tag{10}$$

where the sub-indices {i, j, k} denote the matrix and vector components. A closed-form solution to calculate $\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is provided in Murray et al. (1994).

3.5 Gravitational co-vector

The gravitational co-vector $\boldsymbol{g}(\boldsymbol{q})$ results from the gravitational potential energy $U(\boldsymbol{q}) = \sum_{i=1}^n m_i\ g\ h_i(\boldsymbol{q})$. Here, $h_i(\boldsymbol{q}) \in \mathbb{R}$ is the height of the center of mass of the respective body, $m_i \in \mathbb{R}$ is the body mass, and $g \in \mathbb{R}$ is the gravitational constant. To express the gravitational potential energy $U(\boldsymbol{q})$, the height $h_i(\boldsymbol{q})$ can be derived by using the product of Exponentials Formula (Section 3.1). The gravitational co-vector $\boldsymbol{g}(\boldsymbol{q})$ can be expressed by (Siciliano et al., 2010; Murray et al., 1994):

$$\boldsymbol{g}(\boldsymbol{q}) = -\frac{\partial U(\boldsymbol{q})}{\partial \boldsymbol{q}}. \tag{11}$$

Equation (11) can be implemented by using recursive methods. Another simple geometric way is to express a gravitational wrench $\boldsymbol{F}_i \in \mathbb{R}^6$ with respect to {S} for each body. If the z-direction of {S} coincides with direction of gravity:

$$\boldsymbol{g}(\boldsymbol{q}) = -\sum_{i=1}^{n} {}^H\boldsymbol{J}(\boldsymbol{q})_i^T\ \boldsymbol{F}_i, \tag{12}$$

with ${}^{H}\boldsymbol{J}(\boldsymbol{q})_i$ being the Hybrid Jacobian matrix (Section 3.3) and $\boldsymbol{F}_i = (0, 0, -m_i\ g, 0, 0, 0)^T$.

Part II: Implementation

4 Solving the current challenges

In Part I, we presented the theory of our method. In Part II, we present our MATLAB-based robotics library **Exp[licit]™(EXP)**—the software implementation of Part I. We show that EXP can easily solve the challenges described in Section 2. Methods and detailed MATLAB code examples are presented. All of the mathematical concepts and notations are defined in Part I. Hence, for Part II, we assume that the readers are familiar with the materials of Part I. We compare the computation time of EXP with RVC (Corke and Khatib, 2011), an open-source MATLAB simulation platform based on recursive Newton-Euler algorithms that use DH-convention. We show how EXP can be used for open-kinematic chain mechanisms and provide code examples for the user to try out.

4.1 Rapid prototyping — Exchanging links during design phase

While the traditional approach may have difficulties changing the joint type of the robot, the problem can be easily solved using EXP. All we need is to adapt the respective joint twists $\boldsymbol{\eta}_i$ which is defined at the initial configuration.

To give an example, consider a 4-DOF planar robot with four revolute joints (Figure 5). Assume we want to change the third joint of the robot from revolute to prismatic. With our approach, all that is required is to change the third column of the joint twist array:

$$\begin{bmatrix} | & | & | & | \\ \boldsymbol{\eta}_1 & \boldsymbol{\eta}_2 & \boldsymbol{\eta}_3 & \boldsymbol{\eta}_4 \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -l & -2l & -3l \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} | & | & | & | \\ \boldsymbol{\eta}_1 & \boldsymbol{\eta}_2 & \boldsymbol{\eta}_{3,new} & \boldsymbol{\eta}_4 \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -l & 0 & -3l \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

This can be achieved using the `switchJoint`-method of the `RobotPrimitive`-class. Let `robot` be the 4-DOF planar robot object of `SnakeBot`-class, which is the child class of `RobotPrimitive`-class. The third joint of `robot` can be changed to a prismatic joint by `robot.switchJoint(3, 2, [1;0;0])`. Here, the first argument is the joint number (ordered from proximal to distal) that should be changed. The second argument is the type of joint which should be changed: 1 denotes a revolute joint and 2 denotes a prismatic joint. The third argument is $\hat{\boldsymbol{\omega}}_i$ for a revolute joint, and $\hat{\boldsymbol{v}}_i$ for a prismatic joint. Details are shown in the `main_switchJoint.m` script under the `examples` folder.

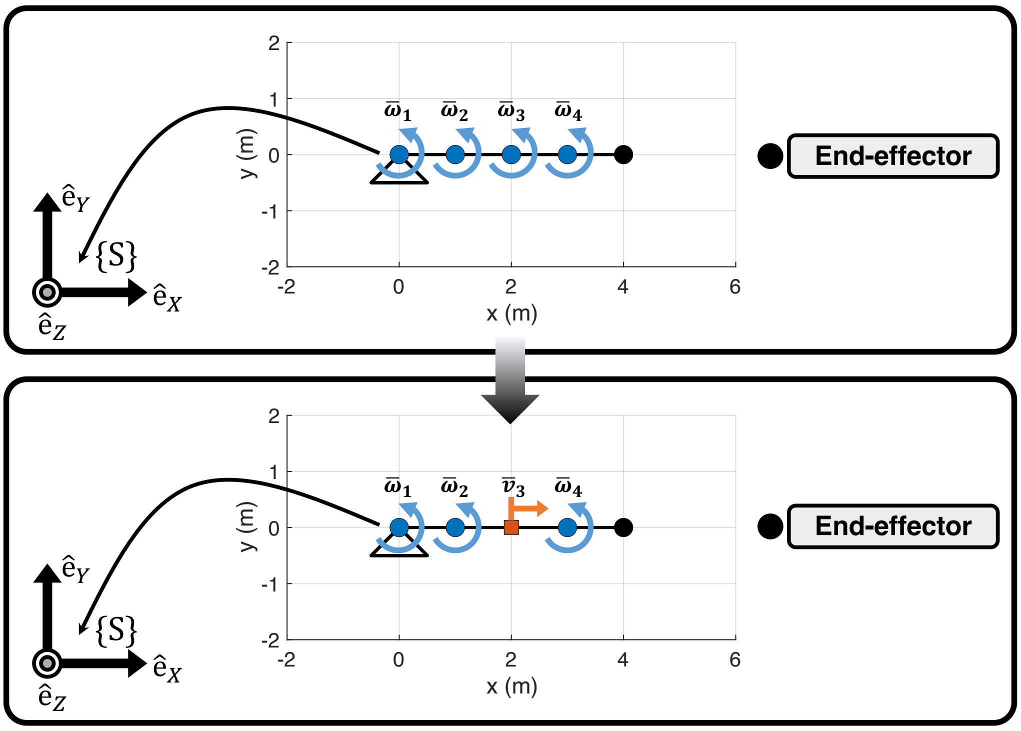

Figure 5: Changing the third revolute joint (blue circle marker) of a 4-DOF planar robot to a prismatic joint (orange square marker). Both robots are at the initial configuration. The coordinate frame {S} is attached to the first joint of the robot. The robot, including the grid and coordinate system, were generated by EXP.

4.2 Adding external kinematics

Consider the robot shown in Glück et al. (2013), where the robot consists of a triple pendulum on a cart. While this 4-DOF robot can be constructed by defining the four joint twists, it can also be constructed by adding two robots with simpler structure—the cart-pole robot and a double pendulum robot (Figure 6). The triple pendulum on a cart can be constructed by attaching the double pendulum robot to the end-effector of the cart-pole robot.

This can be achieved using the `addKinematics`-method of the `RobotPrimitive`-class. Let `robot1` and `robot2` be the objects constructed from the `CartPole`-class and `DoublePendulum`-class, respectively, and both classes are child classes of `RobotPrimitive`-class. Then, `robot1.addKinematics(robot2)` attaches `robot2` to the end-effector of `robot1`, resulting in a triple pendulum on a cart.

The `addKinematics`-method simply concatenates the joint twists array of `robot1` with those of `robot2`. Before the concatenation, the joint twist arrays of `robot2` are mapped to be expressed with respect to the frame {S} of `robot1` (not with respect to the frame {S} of `robot2`). Details can be found in the `addKinematics`-method of the `RobotPrimitive`-class. This example demonstrates the simplicity of EXP when one wants to construct a new robot by adding multiple kinematic structures.

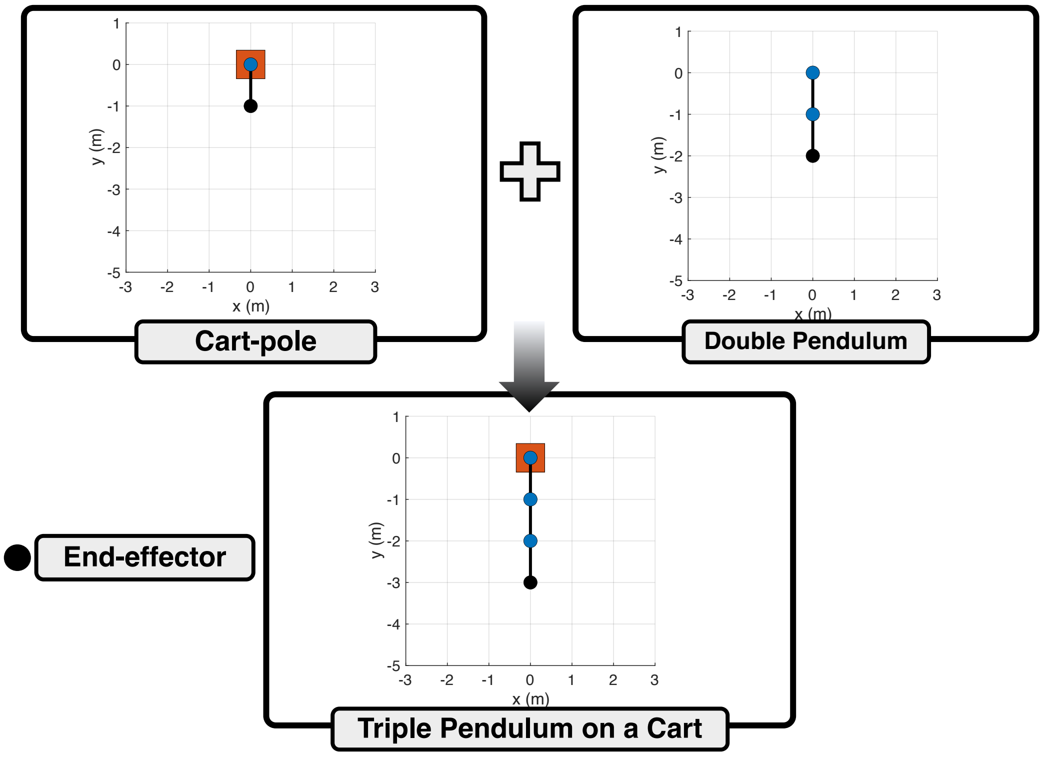

Figure 6: The cart-pole and double pendulum robots, constructed using `CartPole`-class and `DoublePendulum`-class. By attaching the double pendulum to the end-effector of the cart-pole, a triple pendulum on a cart can be created. The images were generated by EXP.

4.3 Changing points of interest on the robot body

Consider the triple pendulum robot on a cart (Figure 7). We assign two tasks to this robot. The first task is to hold the initial position of the end-effector. The second task is to move a point on joint 3 along the x-axis of the stationary coordinate frame.

In `main_triplePendCart.m` under `examples` folder, a simple solution for two superimposed workspace impedance controllers is provided (Hogan, 1985; Hermus et al., 2021). Conceptually, each task is described by a virtual spring-damper-system that pulls the selected point on the robot structure to the desired point. Details about the controller are shown in Appendix B.

To be able to select any point on any robot body, the `getForwardKinematics`-method of the `RobotPrimitive`-class has an optional input. The user can select the desired robot body and a position on this body. The position is expressed with respect to the center of the proximal joint. To calculate the velocity of the center of a body-fixed coordinate frame, the Hybrid Jacobian matrix can be used. The `getHybridJacobian`-method of the `RobotPrimitive`-class has an optional input and the user can specify the body of interest and the position on this body.

The traditional method has to choose a point on the end-effector body to calculate the Forward Kinematics Map. Compared to traditional robot software, the `getForwardKinematics`-method and the `getHybridJacobian`-method can be easily adapted for any point on any robot body since they are derived by the Product of Exponentials Formula (Part I, Section 3.1).

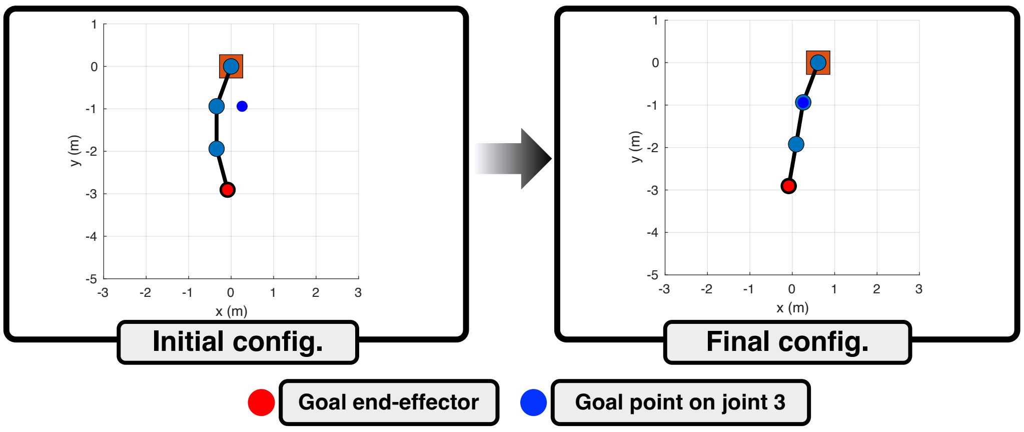

Figure 7: An example to control two points on different robot bodies is provided by the file `main_triplePendCart.m` in EXP. The robot was created by attaching the double pendulum to the end-effector of the cart-pole (Section 4.2).

5 Computational efficiency

The computational efficiency of EXP was compared with the second edition of the RVC MATLAB software (Corke and Khatib, 2011). Version RTB10+MVTB4 (2017) was used for RVC.[5]

Method

Five comparisons were performed:

- Comparison 1: Computation of the Homogeneous transformation matrix, ${}^{S}\boldsymbol{H}_{ee}(t)$.
- Comparison 2: Computation of the Hybrid Jacobian matrix, ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$.
- Comparison 3: Computation of the robot mass matrix, $\boldsymbol{M(q)}$.
- Comparison 4: Computation of the robot centrifugal/Coriolis matrix, $\boldsymbol{c(q,\dot{q})}$.
- Comparison 5: Computation of the robot gravitational co-vector, $\boldsymbol{g(q)}$.

We constructed and used an n-DOF planar robot for comparison (Figure 8). For the RVC software, the robot object was constructed from the `SerialLink`-class which consists of n objects of the `Revolute`-class. For EXP, the robot object was constructed from the `SnakeBot`-class. The length l and mass m of each segment was 1 m and 1 kg, respectively (Figure 8). Robots with various DOF were constructed and tested. The test has been performed with a MacBook air (M1 Chip, 16GB Memory), using MATLAB 2022a. The `timeit()` function was used to measure the computation time.

For the RVC software, the calculations of comparison 1-5 were conducted using the `fkine`, `jacob0`, `inertia`, `coriolis`, `gravload` methods of `SerialLink`-class, respectively. The RVC software uses the DH-convention to calculate ${}^{S}\boldsymbol{H}_{ee}(t)$ and ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})$. Moreover, it uses the recursive Newton-Euler method to calculate $\boldsymbol{M(q)}$, $\boldsymbol{c(q,\dot{q})}$, $\boldsymbol{g(q)}$. For the EXP software, the calculations of comparison 1-5 were conducted using the `getForwardKinematics`, `getHybridJacobian`, `getMassMatrix`, `getCoriolisMatrix`, `getGravityVector` methods of the `RobotPrimitive`-class, respectively. Details of the comparison can be checked in the `main_compare.m` script under `comparisons` directory.

Results

The results of our computational comparisons are shown in Figure 9. For comparison 1, 2 and 3, EXP was faster than the RVC software. For comparison 4 and 5, RVC software was faster than EXP.

For both software solutions, the computation of the Forward Kinematics and the Hybrid Jacobian showed a linear trend. Let us consider a reasonable sample time of 1ms of a simulation software. Within 1ms, the RVC software can compute the Forward Kinematics of the end-effector of planar robot with 15-DOF. For EXP, it needed less than 1ms to compute the Forward Kinematics of 100-DOF robot. For the Hybrid Jacobian, the RVC software needed 5ms for a 20-DOF robot. For EXP, within 5ms it could compute the Hybrid Jacobian of the planar robot with up to 100-DOF.

The computation of the mass matrix showed an exponential trend for both software solutions. For a planar robot with only two joints, the RVC software needed about 3ms. Within this calculation

[5]The software can be downloaded at https://petercorke.com/toolboxes/robotics-toolbox/

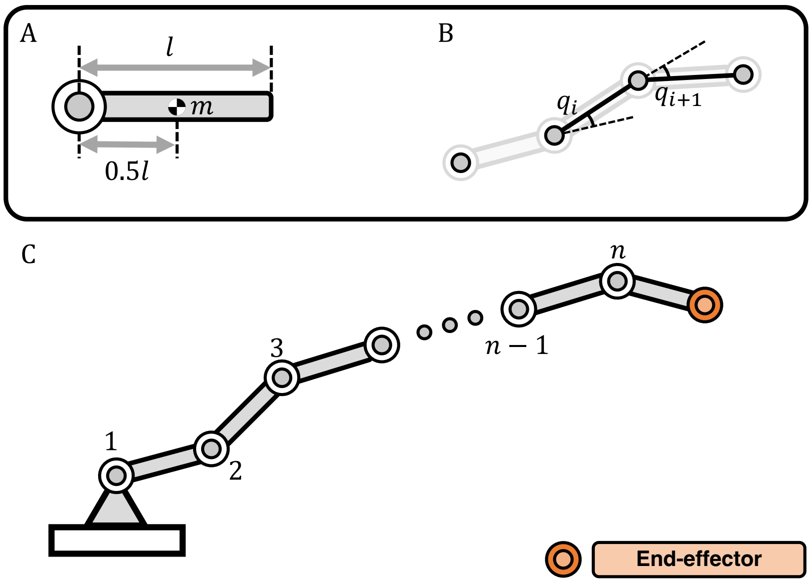

Figure 8: The n-DOF planar robot. (A) Link length and position of the center of mass. (B) Relative angle coordinates. (C) The n-DOF planar robot and its end-effector, drawn in orange color.

time, our software could process the mass matrix of planar robots with up to 15-DOF. For a robot with 6-DOF, EXP needed about 1ms.

For the computation of the centrifugal/Coriolis matrix and the gravitational co-vector of the planar robot, both software showed exponential trend. Compared to the mass matrix, RVC showed better performance than EXP. For more than 15-DOF, the computation of the centrifugal/Coriolis matrix and the gravitational co-vector performed better with RVC. This shows the efficiency of recursive algorithms.

6 Summary

In Part I of the paper, we described some of the current challenges in robotics as seen from a traditional point of view. For the calculation of kinematic and dynamic model parameters, we compared traditional methods with our method based on differential geometry.

The theoretical benefits of the geometric methods shown in Part I were implemented in Part II. We used our EXP software to show how the challenges in robotics, described in Part I of the paper, can be addressed. We demonstrated that EXP can quickly be adapted if changes on the kinematic structures are made, e.g., during the prototype phase of the robot. We showed how new robots can be created by merging simple kinematic structures. With EXP, it is possible to calculate the differential kinematics for any point on any robot body. This is especially useful to control kinematically redundant robots, e.g., humanoid robots. Since we use profound geometric methods, the EXP software is less computational expensive for the computation of Forward Kinematics, Hybrid Jacobian, and mass matrix of the robot.

Summarized, the reason for the simplicity of our software are:

- Simplified parameterization of the robotic system: Joint twists in starting configuration instead

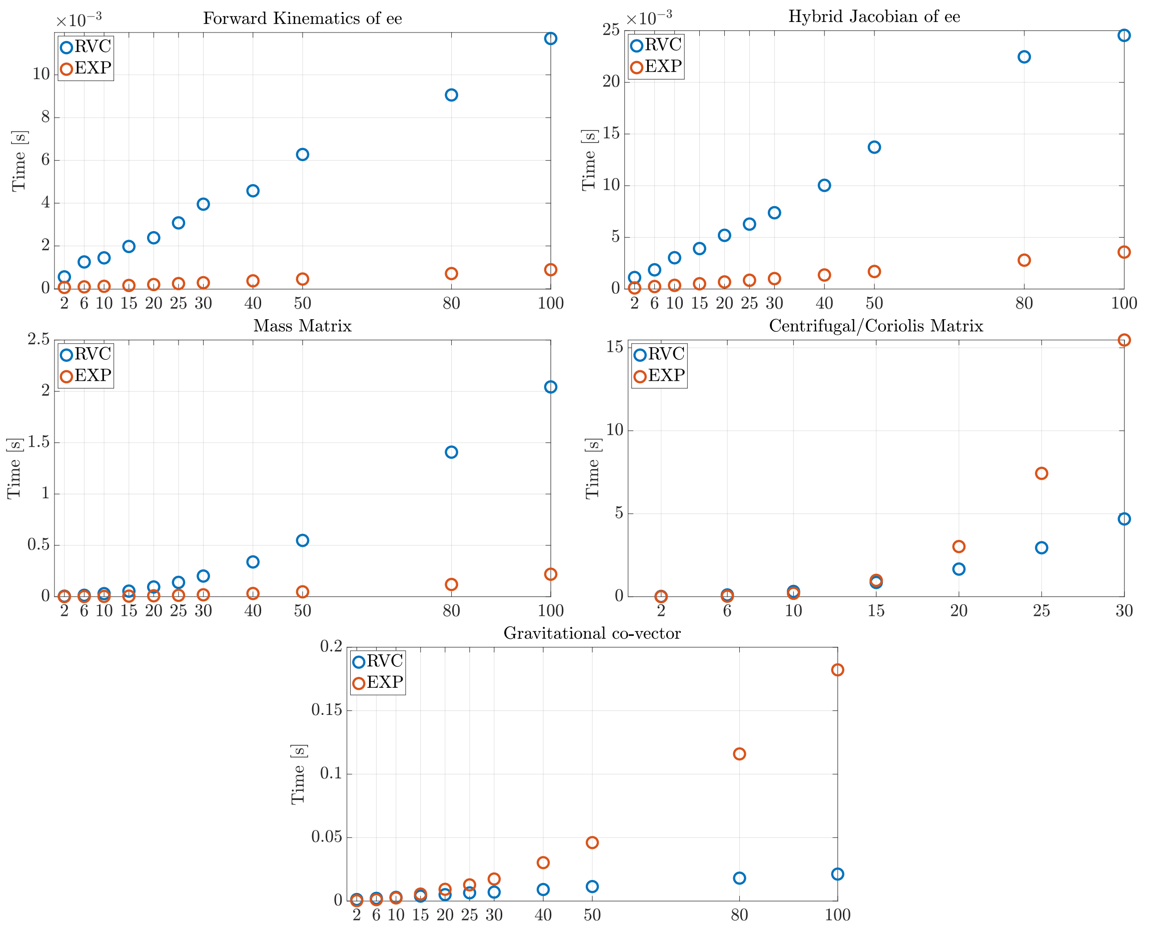

Figure 9: Computation time for five comparisons. Comparison 1: Forward Kinematics of the end-effector (ee); Comparison 2: Hybrid Jacobian of ee; Comparison 3: Mass matrix of the robot. Comparison 4: Centrifugal/Coriolis Matrix of the robot. Comparison 5: Gravitational co-vector of the robot. For comparison 4, data from robots with less than or equal to 30-DOF are plotted.

of four DH-parameters per joint.

- Less coordinate frames for kinematic modelling: One spatial and one body-fixed coordinate frame instead of n-times coordinate frames.
- Uniform calculation for actions on SE(3) instead of separating translational and rotational sub-groups.
- Re-use of joint twists and Exponential maps for the calculation of multiple model components, e.g., Homogeneous transformation matrix, Hybrid Jacobian matrix, and mass matrix.

Finally, we want to mention that the following methods in EXP accept symbolic arguments: `getForwardKinematics`, `getHybridJacobian`, `getMassMatrix`, `getCoriolisMatrix`, and `getGravityVector`. The `RobotPrimitive`-class also accepts a symbolic argument. Hence, EXP supports users who want to derive the symbolic form of the equations of motion of the robot.

7 Limitations and further work

In this paper, we provided the theory and implementation of the equation of motion for an open-kinematic chain manipulator. For closed-kinematic chain manipulators, i.e., parallel mechanisms, the derivation of the equation of motion is more challenging in general. Usually, the Homogeneous transformation matrix and the Jacobian matrix cannot be calculated with a closed-form solution (Siciliano and Khatib, 2007). A parallel mechanism can be viewed as multiple open-kinematic chain mechanisms that are connected to the robot end-effector. The individual open-kinematic chains induce kinematic constraints between each other. The Forward Kinematic Map for a parallel robot with two open-kinematic chain representations can be formulated by (Murray et al., 1994):

$$\begin{aligned} {}^{S}\boldsymbol{H}_{ee}(t) &= \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{1}1}} q_{\mathbf{1}1}\right)\ \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{1}2}} q_{\mathbf{1}2}\right) \cdots \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{1}n}} q_{\mathbf{1}n}\right)\ {}^{S}\boldsymbol{H}_{ee}(t_0) \\ &= \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{2}1}} q_{\mathbf{2}1}\right)\ \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{2}2}} q_{\mathbf{2}2}\right) \cdots \exp\left(\widetilde{\boldsymbol{\eta}_{\mathbf{2}n}} q_{\mathbf{2}n}\right)\ {}^{S}\boldsymbol{H}_{ee}(t_0) \end{aligned} \tag{13}$$

Here, the bold sub-indexes denote the index of the open-kinematic chain. How the kinematic constraints can be formulated depends on the kinematic structure of the parallel mechanism. It can be challenging to identify non-actuated joints such that they satisfy Equation (13). One promising method to formulate the constraint equations is based on screw theory (Li et al., 2019). Since screws are the geometrical representation of Lie algebra and Exponential Maps, the theory of Part I of this paper can also be helpful to analyze closed-kinematic chain mechanisms.

In Part II, we compared EXP with RVC to show that the geometric methods of EXP reduced the computation time for the Forward Kinematics, Hybrid Jacobian and mass matrix of the robot. However, the computation time of EXP for the centrifugal/Coriolis matrix and gravitational co-vector showed opposite trend. The reason is that RVC uses recursive Newton-Euler methods to calculate these matrices. The EXP software is based on closed form solutions which are generally slower. Future work will improve the computation time of the centrifugal/Coriolis matrix and the gravitational co-vector with EXP, e.g., by implementing recursive algorithms.

As can be seen for the example of the mass matrix, Coriolis/centrifugal matrix and gravitational co-vector, and the computation time of EXP increased exponentially when the numbers of degrees of freedom were increased (Figure 9). Featherstone (2016) reduced this computation time to $O(n)$ by using recursive algorithms. It was shown in Stelzle et al. (1995) and discussed in Mueller (2018) that expressing the manipulator dynamics with respect to the body-fixed coordinate frame requires even less computational effort than the solution of Featherstone (2016).

The current version of EXP uses `.m`-MATLAB scripts. To make a fair computational comparison, `.m`-MATLAB scripts were also used for the RVC-method. However, the RVC software provides an option to invoke `MEX`-files to improve the computation speed. `MEX`-files are native `C` or `C++` files that are dynamically linked directly into the MATLAB application at runtime. We discovered that the `MEX`-file option showed better results than our EXP software for the mass matrix computation. In a future version, we will implement `MEX`-files to further improve the computational efficiency of EXP.

So far, the purpose of our software is to enable non-experts to implement robots in simulation. In future work, we will implement the software in C++ and use it for real-time control of robots, e.g., for torque control with the KUKA LBR iiwa (Schreiber et al., 2010).

Appendices

A Nomenclature

Table 1: Nomenclature throughout the paper

Notation and Dimension	Name
$\boldsymbol{q}(t) \in \mathcal{Q}$	Curve on manifold $\mathcal{Q}$
$\boldsymbol{H}(t) \in SE(3)$	Curve on manifold $SE(3)$
${}^{S}\boldsymbol{H}_{ee} \in SE(3)$	Homogeneous transformation of {ee} with respect to {S}
${}^{S}\boldsymbol{R}_{ee} \in SO(3)$	Rotation matrix of {ee} with respect to {S}
$\dot{\boldsymbol{q}} \in \mathbb{R}^n$	Joint velocity
${}^{S}\boldsymbol{\xi} \in se(3)$	Spatial twist
${}^{B}\boldsymbol{\xi} \in se(3)$	Body twist
${}^{S}\boldsymbol{\eta} \in se(3)$	Joint twist
$\hat{\boldsymbol{\omega}}_i \in \mathbb{R}^3$	Unit-axis of rotation
$\hat{\boldsymbol{v}}_i \in \mathbb{R}^3$	Unit-axis of translation
${}^{S}\boldsymbol{p}_{\eta_i} \in \mathbb{R}^3$	Point on twist axis
$e^{\widetilde{\boldsymbol{\eta}_i}\boldsymbol{q}_i} : se(3) \mapsto SE(3)$	Exponential form of joint twist
${}^{S}\boldsymbol{J}(\boldsymbol{q}) : \mathbb{R}^n \mapsto se(3)$	Spatial Jacobian matrix
$\boldsymbol{\eta}' \in se(3)$	Joint twist, column of ${}^{S}\boldsymbol{J}(\boldsymbol{q})$
${}^{B}\boldsymbol{J}(\boldsymbol{q}) : \mathbb{R}^n \mapsto se(3)$	Body Jacobian matrix
$\boldsymbol{\eta}^{\dagger} \in se(3)$	Joint twist, column of ${}^{B}\boldsymbol{J}(\boldsymbol{q})$
${}^{H}\boldsymbol{J}(\boldsymbol{q}) : \mathbb{R}^n \mapsto \mathbb{R}^6$	Hybrid Jacobian matrix
${}^{S}\boldsymbol{V}_{ee} \in \mathbb{R}^6$	Spatial velocity
${}^{S}\boldsymbol{v}_{ee} \in \mathbb{R}^3$	Linear velocity
${}^{S}\boldsymbol{\omega} \in \mathbb{R}^6$	Angular velocity
$\boldsymbol{Ad}_H : se(3) \mapsto se(3)$	Adjoint map
$\boldsymbol{Ad}_H^{-1} : se(3) \mapsto se(3)$	Inverse Adjoint map
$m_i \in \mathbb{R}$	Mass of body i
$\boldsymbol{\mathcal{I}}_i \in \mathbb{R}^{3\times 3}$	Inertia matrix of body i
$\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n\times n}$	Robot mass matrix
$\boldsymbol{\mathcal{M}}_i \in \mathbb{R}^{6\times 6}$	Generalized Inertia Matrix
$T \in \mathbb{R}$	Kinetic energy
$U \in \mathbb{R}$	Potential energy
$\boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{n\times n}$	Coriolis/centrifugal matrix
$\boldsymbol{g}(\boldsymbol{q}) \in \mathbb{R}^n$	Gravitational force
$\boldsymbol{F}_i \in \mathbb{R}^n$	Gravitational wrench of body i

B Impedance superposition of two workspace tasks

Let $\boldsymbol{K}_{ee} \in \mathbb{R}^{3\times 3}$ and $\boldsymbol{K}_3 \in \mathbb{R}^{3\times 3}$ be the diagonal stiffness matrices which represent the virtual springs. The virtual spring of task 1 will be placed between the position $\boldsymbol{p}_{ee}$ on the end-effector body (sub-script ee) and the desired equilibrium position $\boldsymbol{p}_{0,ee}$, which is equal to the initial position of $\{ee\}$. The virtual spring of task 2 will be placed between the position $\boldsymbol{p}_3$ on joint 3 (sub-script 3) and the desired equilibrium position $\boldsymbol{p}_{0,3}$. Here, $\boldsymbol{p}_{0,3}$ is regulated by a simple trajectory generator in order for the point on joint 3 to move along the x-axis of the stationary coordinate frame.

To guarantee asymptotic convergence, virtual dampers are introduced. The resistive elements $\boldsymbol{B}_{ee} \in \mathbb{R}^{3\times3}$ and $\boldsymbol{B}_3 \in \mathbb{R}^{3\times3}$ act on the linear velocities $\dot{\boldsymbol{p}}_{ee} \in \mathbb{R}^3$ and $\dot{\boldsymbol{p}}_3 \in \mathbb{R}^3$, respectively.

With the transpose of the Hybrid Jacobian matrices ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})_{ee} \in \mathbb{R}^{3\times4}$ and ${}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})_3 \in \mathbb{R}^{3\times4}$, the task forces can be mapped to task torques. To regulate uncontrolled joint motions, a joint damper $\boldsymbol{B}_q \in \mathbb{R}^{4\times4}$ is introduced that acts on the joint motion $\dot{\boldsymbol{q}}$. The final control torque $\boldsymbol{\tau} \in \mathbb{R}^4$ can be calculated by:

$$\boldsymbol{\tau} = \underbrace{{}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})_{ee}^T\big(\boldsymbol{K}_{ee}(\boldsymbol{p}_{0,ee} - \boldsymbol{p}_{ee}) - \boldsymbol{B}_{ee}\ \dot{\boldsymbol{p}}_{ee}\big)}_{\text{Task 1}} + \underbrace{{}^{\mathrm{H}}\boldsymbol{J}(\boldsymbol{q})_3^T\big(\boldsymbol{K}_3(\boldsymbol{p}_{0,3} - \boldsymbol{p}_3) - \boldsymbol{B}_3\ \dot{\boldsymbol{p}}_3\big)}_{\text{Task 2}} - \boldsymbol{B}_q\ \dot{\boldsymbol{q}}$$

For this example, we assume that the gravitational forces as well as Coriolis and centrifugal effects are already compensated.

References

Angeles J (2006) *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms (Mechanical Engineering Series)*. Berlin, Heidelberg: Springer-Verlag. ISBN 0387294120.

Brockett RW (1984) *Robotic manipulators and the product of exponentials formula.* Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-38826-5.

Corke P and Khatib O (2011) *Robotics, Vision and Control - Fundamental Algorithms in MATLAB, Springer Tracts in Advanced Robotics*, volume 73. Springer. ISBN 978-3-642-20143-1.

Craig JJ (1986) *Introduction to robotics : mechanics & control / John J. Craig.* Reading, Mass.: Addison-Wesley Pub. Co.,. ISBN 0201103265.

Denavit J and Hartenberg RS (1955) A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics* 22: 215–221.

Dubrovin B, Burns R, Fomenko A and Novikov S (2013) *Modern Geometry - Methods and Applications: Part I. The Geometry of Surfaces, Transformation Groups, and Fields.* Graduate Texts in Mathematics. Springer New York. ISBN 9781468499469. URL `https://books.google.de/books?id=ZEn0BwAAQBAJ`.

EC-Project (2017) Refills - scenarios and robots (id: 731590). URL `http://www.refills-project.eu/index.php/about/project/scenario-and-robots`. Accessed: 2022-09-12.

EC-Project (2019) Murab - mri and ultrasound robotic assisted biopsy (id: 688188). URL `https://www.murabproject.eu/`. Accessed: 2022-09-12.

El Zaatari S, Marei M, Li W and Usman Z (2019) Cobot programming for collaborative industrial tasks: An overview. *Robotics and autonomous systems.* 116: 162–180.

Featherstone R (2016) *Rigid Body Dynamics Algorithms.* Springer Publishing Company, Incorporated. ISBN 1489978682.

Felis ML (2016) Rbdl: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots* : 1–17DOI:10.1007/s10514-016-9574-0. URL `http://dx.doi.org/10.1007/s10514-016-9574-0`.

Glück T, Eder A and Kugi A (2013) Swing-up control of a triple pendulum on a cart with experimental validation. *Automatica* 49(3): 801–808.

Hermus J, Lachner J, Verdi D and Hogan N (2021) Exploiting redundancy to facilitate physical interaction. *IEEE Transactions on Robotics* 38(1): 599–615.

Hjorth S, Lachner J, Stramigioli S, Madsen O and Chrysostomou D (2021) An energy-based approach for the integration of collaborative redundant robots in restricted work environments. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 7152–7158. DOI:10.1109/IROS45743.2020.9341561.

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

KUKA and Daimler (2019) Corrosion protection and leak-tightness. URL `https://www.kuka.com/en-de/industries/solutions-database/2019/09/seam-sealant-application-at-daimler`. Accessed: 2022-09-12.

Lachner J (2022) *A geometric approach to robotic manipulation in physical human-robot interaction.* PhD Thesis, University of Twente, Netherlands. DOI:10.3990/1.9789036553568.

Lachner J, Allmendinger F, Stramigioli S and Hogan N (2022) Shaping impedances to comply with constrained task dynamics. *IEEE Transactions on Robotics* : 1–18DOI:10.1109/TRO.2022.3153949.

Levi E (1905) Sulla struttura dei gruppi finiti e continui. *Atti della Reale Accademia delle scienze di Torino* 40: 551–565.

Li Q, Hervé J and Ye W (2019) *Geometric Method for Type Synthesis of Parallel Manipulators.* Springer Tracts in Mechanical Engineering. Springer Singapore. ISBN 9789811387555.

Mueller A (2017) Recursive second-order inverse dynamics for serial manipulators. In: *2017 IEEE International Conference on Robotics and Automation (ICRA).* pp. 2483–2489. DOI:10.1109/ICRA.2017.7989289.

Mueller A (2018) Screw and lie group theory in multibody dynamics. *Multibody System Dynamics* 42: 219–248. DOI:https://doi.org/10.1007/s11044-017-9583-6.

Mueller A (2021) An o(n)-algorithm for the higher-order kinematics and inverse dynamics of serial manipulators using spatial representation of twists. *IEEE Robotics and Automation Letters* 6(2): 397–404. DOI:10.1109/LRA.2020.3044028.

Murray R, Li Z, Sastry S and Sastry S (1994) *A Mathematical Introduction to Robotic Manipulation.* Taylor & Francis. ISBN 9780849379819.

Park F (1994) Computational aspects of the product-of-exponentials formula for robot kinematics. *IEEE transactions on automatic control.* 39(3): 643–647.

Schreiber G, Stemmer A and Bischoff R (2010) The Fast Research Interface for the KUKA Lightweight Robot. *IEEE ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications* 1(1).

Siciliano B and Khatib O (2007) *Springer Handbook of Robotics.* Berlin, Heidelberg: Springer-Verlag. ISBN 354023957X.

Siciliano B, Sciavicco L, Villani L and Oriolo G (2010) *Robotics: Modelling, Planning and Control.* Springer Publishing Company, Incorporated. ISBN 1849966346.

Stelzle W, Kecskeméthy A and Hiller M (1995) A comparative study of recursive methods. *Archive of Applied Mechanics* 66(1): 1432–0681. DOI:10.1007/BF00786685.

Stramigioli S (2022) Modern robotics. URL `https://vimeo.com/102509138`. Course module 191211060.

Stramigioli S and Bruyninckx H (2001) Tutorial: Geometry and screw theory for robotics. In: *2001 IEEE International Conference on Robotics and Automation (ICRA).*

Welleweerd M (2022) *Robot-assisted biopsies on MR-detected lesions.* PhD Thesis, University of Twente, Netherlands. DOI:10.3990/1.9789036553223.

Zumpe V, Huesing M, Horeis J and Corves B (2019) Pre-processing for task-based kinematic synthesis. DOI:10.17185/duepublico/48194.