A numeric derivation for fast regressive modeling of manipulator dynamics

Steffan Lloyd[a,*], Rishad Irani[a], Mojtaba Ahmadi[a]

[a]*Carleton University, 1125 Colonel By Dr, Ottawa, Ontario, CA. K1S 5B6*

Abstract

We present *Pseudo-Symbolic Dynamic Modeling* (PSDM), a novel method of deriving the closed-form equations of motion of a serial kinematic chain, using base inertial parameters. PSDM is a numerical algorithm, yet allows for model simplification and pre-computation generally only possible using symbolic software. In PSDM, we characterize the form of the dynamic equations to build a set of functions guaranteed to form a linear basis for the inverse dynamics function of the manipulator. Then, a two-step numerical analysis is performed to reduce this set into a minimal dynamic model, in regressor form. PSDM offers a fast and procedural method of generating simplified dynamic models. Extensions to the algorithm allow for fast real-time code generation, forward dynamic modeling, and increased model efficiency through the elimination of minimally important model elements. The algorithm is benchmarked on common robot configurations and shown to be attractive for the dynamic modeling of up to 7-DOF manipulators, in terms of derivation time and real-time evaluation speed. Additionally, a MATLAB implementation of the algorithm has been developed and is made available for general use.

Keywords: multi-body dynamics; dynamic calibration; robot dynamics; computational dynamics.

1. Introduction

Dynamic modeling of an n-link serial rigid-body chain, representing a robotic manipulator or other mechanical system, is an important subject which has received thorough treatment in literature, particularly in the 1980s and 1990s. Dynamic models are an important tool for many control algorithms and have applications for feed-forward dynamic compensation [1, 2], adaptive control [3–6], state estimation algorithms [7], motion planning [8, 9], manipulator design [10, 11] and much more. A dynamic model is commonly represented with the matrix differential equation

$$\boldsymbol{\tau} = \mathbf{D}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}), \tag{1}$$

where $\mathbf{D}(\mathbf{q})$ is the *mass matrix*, $\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}$ is a vector of Coriolis and centrifugal terms, and $\mathbf{G}(\mathbf{q})$ is the *gravity vector*. These models are implicitly functions of the robot's inertial parameters, denoted as an $n \times 10$ matrix

$$\mathbf{X} = \begin{bmatrix} m_1 & r_{x,1} & r_{y,1} & r_{z,1} & I_{xx,1} & I_{yy,1} & I_{zz,1} & I_{xy,1} & I_{xz,1} & I_{yz,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_n & r_{x,n} & r_{y,n} & r_{z,n} & I_{xx,n} & I_{yy,n} & I_{zz,n} & I_{xy,n} & I_{xz,n} & I_{yz,n} \end{bmatrix}, \tag{2}$$

containing, for each link i, the mass m_i, the center of gravity coordinates $r_{x,i}$, $r_{y,i}$, $r_{z,i}$, and the six components of the inertia tensor $I_{xx,i}$, $I_{yy,i}$, $I_{zz,i}$, $I_{xy,i}$, $I_{xz,i}$, and $I_{yz,i}$. For commercial and industrial robots, these parameters are generally unknown, and without disassembling the manipulator, it is difficult to estimate them experimentally. Without accurate knowledge of the inertial parameters $\mathbf{X}$, dynamic models are of limited accuracy and cannot be leveraged properly. A more useful form of the robot dynamics is the *regressor form*, where one exploits the linearity of (1) with respect to $\mathbf{X}$ to rearrange the equation to

$$\boldsymbol{\tau} = \mathbf{Y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\boldsymbol{\theta}_b, \tag{3}$$

*Corresponding author
Email address: Steffan.Lloyd@carleton.ca (Steffan Lloyd)

where $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is the $n \times \ell$ *manipulator regressor* and $\boldsymbol{\theta}_b$ is an ℓ-vector of *base inertial parameters*. Here, $\boldsymbol{\theta}_b$ contains all the inertial information necessary to calibrate dynamic model, but has fewer than the $10n$ elements of $\mathbf{X}$ due to constrained motion in the joints and coupling between links. In this form, the parameter vector $\boldsymbol{\theta}_b$ can be identified offline using well-established system identification techniques, or online in adaptive controller designs such as those proposed by Slotine and Li [3], Craig et al. [4], and many more.

Much research has been performed to determine the best way to derive the equations for $\mathbf{Y}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ in a computationally efficient manner. For a long time, the symbolic derivation of dynamic models was done through the symbolic Lagrange formulation [12, 13]. However, since then, significant work has been done on reducing the computational requirements of the dynamic model. In 1980, Kane presented an alternative symbolic method, which came to be known as Kane's method [14], and which allowed for easier derivation and fewer computations compared to the Lagrange method. Also in 1980, the *recursive Newton-Euler* (RNE) method, proposed by Luh et al. [15], was introduced. The RNE method is a recursive algorithm with linear complexity $O(n)$. This complexity is a significant improvement over the Lagrange formulation, which had $O(n^4)$ complexity [16]. A recursive form of the Lagrange formulation has also been proposed by Hollerbach [16]. However, none of these formulations were given in regressor form, making proper dynamic model calibration difficult. Variants have since been proposed to the RNE method, which do return a regressor matrix, such as in the work of Atkeson et al. [17] or Khalil and Kleinfinger [18]. Additionally, rules for identifying the base inertial parameters were developed by Mayeda et al. [19], Gautier [20], and others, and integrated into symbolic formulations of the dynamic models [18, 21].

The existing work can be categorized into two main subsets. Numerical methods, such as those based on the RNE algorithm, are fully procedural algorithms taking as input some kinematic parameters (such as a DH table) and inertial parameters. These methods are simple to use since they need only be programmed once and can subsequently be used with any manipulator, given only some set of kinematic and inertial parameters. However, they may perform redundant computations since they do not take advantage of any possible simplifications of the dynamic model, which arise due to alignment in the mechanical system structure or due to zero-valued kinematic and inertial parameters. Symbolic algorithms, conversely, leverage symbolic simplification routines to eliminate redundant terms and simplify the calculations, resulting in a lower computational complexity [22]. These methods are more difficult to use than the numerical methods since a derivation step is required before the model can be used. As the dynamic models of a manipulator are exceedingly complex, this derivation can be quite lengthy. Moreover, common symbolic manipulation software such as *Maple* or *Mathematica* are proprietary and require expensive licenses to use, precluding their use for many. These limitations create an accessibility problem for research and development. Some open-source symbolic software does exist, such as *SymPy* [23], as well as robotics specific symbolic modeling software [24, 25]. These toolboxes are useful, but often have limited support for computing the regressor using base inertial parameters. Integration of symbolically-derived models into robotic software packages is also difficult, since they must have access to symbolic software to perform the necessary derivations. Symbolic routines, in general, can also lack robustness, since they rely on simplification algorithms which use significant memory and can fail or give suboptimal results.

This paper introduces a novel method of derivation for dynamic models of robotic manipulators and other serial rigid-body linkages, which we name *Pseudo-Symbolic Dynamic Modeling*, or PSDM. The PSDM algorithm is not directly based on either the Lagrange or Newton-Euler formulation, and approaches the problem of dynamic modeling from a different perspective than traditional methods. In PSDM, we define a search space of functions which are guaranteed to span the inverse dynamics function of the robot, $\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. A numerical analysis is then performed, leveraging the numerical recursive Newton-Euler algorithm, to determine which functions in this search space contribute non-trivially to the manipulator dynamic model. A secondary numerical analysis is then done to determine the relationship between these identified functions and the manipulator's inertial parameters. The algorithm results in a *pseudo-symbolic* representation of the function $\boldsymbol{\tau}$, in which the full symbolic equations of (3) are encoded numerically, in matrix form.

The proposed PSDM method gives a convenient middle ground between the numerical and symbolic algorithms. It is a completely numerical algorithm and can be implemented using only basic linear algebra operations. However, PSDM also allows for the same simplifications and reduction in complexity generally only available using symbolic software. This attribute makes PSDM both fast and widely accessible – even to those without high-level expertise in the field. The model returned by PSDM is in regressor form, allowing for easy dynamic calibration or use in adaptive control algorithms. Moreover, the dynamic model resulting from a PSDM derivation is encoded numerically in a very regular matrix form. This regularity allows for symbolic-style manipulations to be performed on the model using linear algebra operations. The model is therefore flexible to use and manipulate, which allows for some convenient extensions such as optimized real-time code generation, forward dynamic modeling, and a complexity reduction technique, which allows for an intuitive trade-off between model accuracy and evaluation time. We note that PSDM is *not* an $O(n)$ algorithm, and as such, does not scale well to large robots. However, for small to medium-sized, revolute or prismatic jointed, rigid body chains ($n < 8$), PSDM can provide a robust and completely procedural means of establishing an efficient forward and inverse dynamic model, in regressor form. Additionally, a full implementation of the algorithm is developed in Matlab code and made available for general use.

This work will give a complete description of the logic, implementation and use of the PSDM algorithm for dynamic modeling and calibration. First, Section 2 will present a full mathematical derivation of the PSDM method and show how it can be used to procedurally generate a numerical dynamic model of an arbitrary serial manipulator, in regressor form. Section 3 will then discuss details relevant to the practical implementation and use of the algorithm. In Section 4, we will describe three extensions to PSDM – in the generation of optimized real-time code, the development of the direct (forward) dynamic model, and how PSDM can be used to allow for a trade-off of computational load and model accuracy. Finally, Section 5 will demonstrate the application of PSDM to some common manipulators and provides benchmarks and comparisons to other existing dynamic modeling routines.

2. Pseudo-Symbolic Dynamic Modeling

This section will give a mathematical overview of the PSDM algorithm. Section 2.1 presents the derivation for the *Pseudo-Symbolic Representation Theorem*, which identifies a well-defined set of functions that are guaranteed to span the inverse dynamics of any serial kinematic chain. Section 2.2 then shows how to use this set of functions to form a minimal dynamic model for the manipulator, in regressor form.

2.1. The Pseudo-Symbolic Representation Theorem

In the following section, we present the *Pseudo-Symbolic Representation Theorem* and associated definitions, with the intention to show that the inverse dynamic function of a manipulator is guaranteed to be in a particular form. More specifically, we wish to show that it is possible to define a set of functions which are guaranteed to be a basis for the torques of each joint of the robot. Throughout the derivation, we will use the canonical example of a two-link planar manipulator to illustrate the process. We begin with a minor extension to the concept of linear dependency of functions, to describe the *entrywise* relationship of a matrix function to a set of scalar functions, which will be used in later discussions.

Definition 1. *A vector or matrix function* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$*, where* $n, m \geq 1$*, we say to be* ***spanned entrywise*** *by a set of functions* $\mathcal{S}$*, if each entry* f_i *in* $\mathbf{f}(\mathbf{x})$*,* $\mathbf{x} \in \mathbb{R}^n$*, can be obtained through a linear combination of functions* $\sigma_j(\mathbf{x}) \in \mathcal{S}$*, as*

$$f_i(\mathbf{x}) = \sum_{j=0}^{k} \kappa_j \sigma_j(\mathbf{x}); \qquad k \in \mathbb{N}, \quad \sigma_j(\mathbf{x}) \in \mathcal{S}, \quad \kappa_j \in \mathbb{R}. \tag{4}$$

To denote this relationship, we will use the modified notation $\mathbf{f}(\mathbf{x}) \in \operatorname{span}^*\{\mathcal{S}\}$.

We now define two sets of functions. These sets describe the trigonometric relationships between adjacent manipulator links for *revolute* and *prismatic* joints, and serve as building blocks for later results.

Definition 2. *The* ***revolute multiplier functions of order k*** *for an angle* λ*, denoted by* $\mathcal{Z}_\lambda^{(k)}$*, is the set of functions obtained through products of non-negative integer powers of* $\sin\lambda$ *and* $\cos\lambda$ *with maximum combined order* k*,*

$$\mathcal{Z}_\lambda^{(k)} = \Big\{ \zeta_i : \mathbb{R} \to [-1, 1] \;\Big|\; \zeta_i(\lambda) = \sin^\alpha\lambda \cdot \cos^\beta\lambda, \;\alpha, \beta \in \mathbb{N}_0, \;\alpha + \beta \leq k \Big\}. \tag{5}$$

$\mathcal{Z}_\lambda^{(k)}$ can be viewed as the product across the elements of all multiset of $\{\sin\lambda, \cos\lambda\}$, with maximum multiplicity of k. Therefore, we can obtain the size of this set from the sum of the multiset coefficients [26] of $0, \ldots, k$ selections from a set of size 2, as

$$\left|\mathcal{Z}_\lambda^{(k)}\right| = \sum_{i=0}^{k} \left(\!\!\binom{2}{i}\!\!\right) = \sum_{i=0}^{k} \binom{2+i-1}{i} = \sum_{i=0}^{k} i + 1 = \big(1 + 2 + \cdots + k\big) + \big(k+1\big) = \frac{(k+1)(k+2)}{2}, \tag{6}$$

where the identity $1 + 2 + \cdots + k = k(k+1)/2$ is used in the final step of simplification.

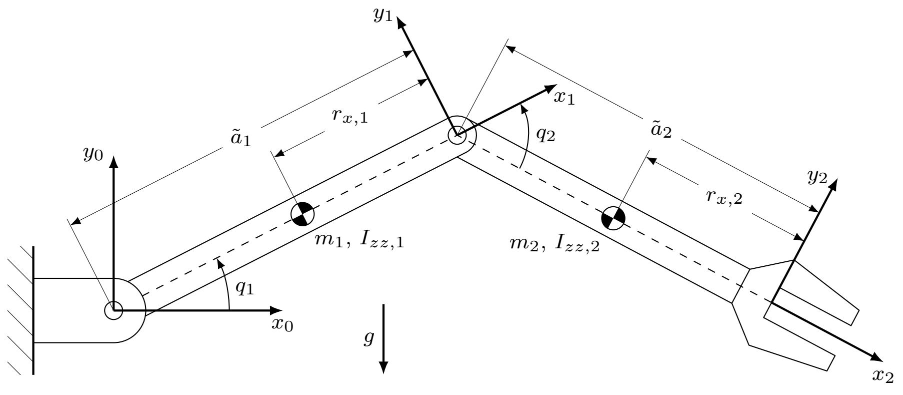


Figure 1: A two-link planar manipulator.

Definition 3. *The* ***prismatic multiplier functions of order k*** *for a variable λ, denoted by $\mathcal{D}_\lambda^{(k)}$, is the set of functions obtained by raising λ to a non-negative integer power less than or equal to k,*

$$\mathcal{D}_\lambda^{(k)} = \Big\{ \delta_i : \mathbb{R} \to \mathbb{R} \ \Big| \ \delta_i(\lambda) = \lambda^\alpha,\ \alpha \in \mathbb{N}_0,\ \ \alpha \le k \Big\}. \tag{7}$$

By inspection, a set $\mathcal{D}_\lambda^{(k)}$ has size $k+1$. It is possible to combine these two sets into another set, which will describe the more general geometric relationship between any two links on a given manipulator.

Definition 4. *For an n-degree of freedom serial manipulator, we define the* ***geometric multiplier functions of order k*** *to be the set $\Upsilon^{(k)}$ of all functions obtained through products of a function taken from the sets $\mathcal{Z}_{q_i}^{(k)}$ or $\mathcal{D}_{q_i}^{(k)}$ for each joint i as*

$$\Upsilon^{(k)} = \Big\{ \upsilon : \mathbb{R}^n \to \mathbb{R} \ \Big| \ \upsilon(\mathbf{q}) = \prod_{i=1}^{n} \sigma_i(q_i);\ \sigma_i \in \mathcal{S}_{q_i}^{(k)} \Big\}, \qquad \mathcal{S}_{q_i}^{(k)} = \begin{cases} \mathcal{Z}_{q_i}^{(k)}, & \textit{if joint } i \textit{ is revolute,} \\ \mathcal{D}_{q_i}^{(k)}, & \textit{if joint } i \textit{ is prismatic.} \end{cases} \tag{8}$$

The size of $\Upsilon^{(k)}$ for an n-link manipulator is

$$\left|\Upsilon^{(k)}\right| = \left|\mathcal{Z}_{q_i}^{(k)}\right|^{n_r} \left|\mathcal{D}_{q_i}^{(k)}\right|^{n_p} = \frac{(k+1)^n (k+2)^{n_r}}{2^{n_r}}, \tag{9}$$

where n_r and n_p are the number of revolute and prismatic joints in the robot, respectively. To illustrate the sets $\mathcal{Z}_{q_i}^{(k)}$, $\mathcal{D}_{q_i}^{(k)}$, and $\Upsilon^{(k)}$ we use the example of a two-link planar robotic arm, as shown in Figure 1.

Example 1. The two-link manipulator in Figure 1 has two revolute joints with generalized joint coordinates q_1 and q_2. Using condensed trigonometric notation ($\cos q_i \triangleq c_i$, $\sin q_i \triangleq s_i$), the sets of angular multiplier functions of order 1 and 2 for both joints are

$$\mathcal{Z}_{q_1}^{(1)} = \big\{1,\ s_1,\ c_1\big\}, \qquad \mathcal{Z}_{q_1}^{(2)} = \big\{1,\ s_1,\ c_1,\ s_1c_1,\ {s_1}^2,\ {c_1}^2\big\},$$
$$\mathcal{Z}_{q_2}^{(1)} = \big\{1,\ s_2,\ c_2\big\}, \qquad \mathcal{Z}_{q_2}^{(2)} = \big\{1,\ s_2,\ c_2,\ s_2c_2,\ {s_2}^2,\ {c_2}^2\big\}.$$

The set of geometric multiplier functions of order 1 of the manipulator consists of the nine functions

$$\Upsilon^{(1)} = \big\{\, 1,\ c_1,\ c_2,\ s_1,\ s_2,\ s_1s_2,\ s_1c_2,\ c_1s_2,\ c_1c_2 \,\big\},$$

and the set of geometric multiplier functions of order 2 consists of the 36 functions

$$\Upsilon^{(2)} = \big\{ \begin{array}{llllllllllll} 1, & c_1c_2, & c_1{c_2}^2, & s_1{s_2}^2, & {c_1}^2{c_2}^2, & c_1c_2s_2, & c_1, & c_1s_1, & {c_1}^2c_2, & {s_1}^2s_2, & {c_1}^2{s_2}^2, & c_1s_1s_2, \\ c_2, & c_1s_2, & c_1{s_2}^2, & {c_1}^2, & {c_2}^2{s_1}^2, & c_2s_1s_2, & s_1, & c_2s_1, & c_2{s_1}^2, & {c_2}^2, & c_1s_1{s_2}^2, & c_1{c_2}^2s_1, \\ s_2, & c_2s_2, & {c_1}^2s_2, & {s_1}^2, & c_2{s_1}^2s_2, & {c_1}^2c_2s_2, & {s_1}^2{s_2}^2, & s_1s_2, & {c_2}^2s_1, & {s_2}^2, & c_1c_2s_1, & c_1c_2s_1s_2 \big\}. \end{array} \qquad \triangleleft$$

The set of geometric multiplier functions $\Upsilon^{(k)}$ for a manipulator describes many of the geometric relations between the joints of the robot. More specifically, we will show over the following sections that these sets of geometric functions *entrywise span* the kinematic transformation matrices and the Jacobian matrices of the robot. First, however, we show two properties of functions in the entrywise span of the sets $\Upsilon^{(k)}$, under multiplication and derivation. These properties will be necessary for the proofs which follow.

Proposition 1. *The matrix multiplication of two vector or matrix functions* $\mathbf{f}_1(\mathbf{q}) \in \operatorname{span}^*\{\Upsilon^{(k_1)}\}$ *and* $\mathbf{f}_2(\mathbf{q}) \in \operatorname{span}^*\{\Upsilon^{(k_2)}\}$ *will be entrywise spanned by the set* $\Upsilon^{(k_1+k_2)}$.

Proof. For any two *revolute multiplier* functions $\zeta_1 \in \mathcal{Z}_\lambda^{(k_1)}$, $\zeta_2 \in \mathcal{Z}_\lambda^{(k_2)}$, their product ζ_3 is

$$\zeta_3(\lambda) = \zeta_1(\lambda) \cdot \zeta_2(\lambda) = \sin(\lambda)^{\alpha_1} \cos(\lambda)^{\beta_1} \sin(\lambda)^{\alpha_2} \cos(\lambda)^{\beta_2} = \sin(\lambda)^{\alpha_1+\alpha_2} \cos(\lambda)^{\beta_1+\beta_2}, \tag{10}$$

where $\zeta_3 \in \mathcal{Z}_\lambda^{(k_1+k_2)}$ by definition, since $(\alpha_1 + \beta_1) + (\alpha_2 + \beta_2) \leq k_1 + k_2$. Similarly, for any two *prismatic multiplier* functions $\delta_1 \in \mathcal{D}_\lambda^{(k_1)}$, $\delta_2 \in \mathcal{D}_\lambda^{(k_2)}$, their product δ_3 is

$$\delta_3(\lambda) = \delta_1(\lambda) \cdot \delta_2(\lambda) = \lambda^{\alpha_1}\lambda^{\alpha_2} = \lambda^{\alpha_1+\alpha_2} \tag{11}$$

where $\delta_3 \in \mathcal{D}_\lambda^{(k_1+k_2)}$ since $\alpha_1 + \alpha_2 \leq k_1 + k_2$. Let the symbol σ represent a function from $\mathcal{Z}$ or $\mathcal{D}$, as applicable. Then, for any two functions $\upsilon_1(\mathbf{q}) \in \Upsilon^{(k_1)}$, $\upsilon_2(\mathbf{q}) \in \Upsilon^{(k_2)}$, we have

$$\upsilon_3(\mathbf{q}) = \upsilon_1(\mathbf{q}) \cdot \upsilon_2(\mathbf{q}) = \sigma_{1,1}(q_1) \cdots \sigma_{1,n}(q_n) \cdot \sigma_{2,1}(q_1) \cdots \sigma_{2,n}(q_n) = \sigma_{3,1}(q_1) \cdots \sigma_{3,n}(q_n) \tag{12}$$

where $\sigma_{3,i} \in \mathcal{Z}_{q_i}^{(k_1+k_2)}$ if i is a revolute joint, by (10), and $\sigma_{3,i} \in \mathcal{D}_{q_i}^{(k_1+k_2)}$ if i is a prismatic joint, by (11). Then, by definition, $\upsilon_3(\mathbf{q}) \in \Upsilon^{(k_1+k_2)}$.

When the two functions $\mathbf{f}_1(\mathbf{q})$ and $\mathbf{f}_2(\mathbf{q})$ are multiplied, each of the resulting matrix elements will be a linear combination of products of one element from each of the original matrices. Each of these elements from the original matrices are, in turn, linear combinations of the functions $\Upsilon^{(k_1)}$ and $\Upsilon^{(k_2)}$. Therefore, after expanding the multiplications, the result will be a linear combination of products of one function from each of the sets $\Upsilon^{(k_1)}$ and $\Upsilon^{(k_2)}$ as in (12). Each of the elements of the resulting matrix will therefore be spanned by the set $\Upsilon^{(k_1+k_2)}$. Thus, by definition, the entire matrix will be entrywise spanned by the same set. □

Proposition 2. *The set of functions which are entrywise spanned by* $\Upsilon^{(k)}$ *is closed under differentiation by* q_i*, for* $i = 1, \ldots, n$.

Proof. For any *revolute multiplier* function $\zeta \in \mathcal{Z}_\lambda^{(k)}$, we have

$$\frac{d}{d\lambda}\zeta(\lambda) = \frac{d}{d\lambda}\sin^\alpha(\lambda)\cos^\beta(\lambda) = \alpha\cos(\lambda)^{\beta+1}\sin(\lambda)^{\alpha-1} - \beta\cos(\lambda)^{\beta-1}\sin(\lambda)^{\alpha+1} = \alpha\,\zeta_1(\lambda) - \beta\,\zeta_2(\lambda),$$

where $\zeta_1, \zeta_2 \in \mathcal{Z}_\lambda^{(k)}$ by definition, since $(\beta+1)+(\alpha-1) = (\beta-1)+(\alpha+1) = \alpha+\beta \leq k$, and thus the derivative of any function in $\mathcal{Z}_\lambda^{(k)}$ w.r.t. λ is spanned by $\mathcal{Z}_\lambda^{(k)}$. Similarly, for any *prismatic multiplier function* $\delta \in \mathcal{D}_\lambda^{(k)}$,

$$\frac{d}{d\lambda}\delta(\lambda) = \frac{d}{d\lambda}\lambda^\alpha = \begin{Bmatrix} \alpha\,\lambda^{\alpha-1} & \text{if } \alpha \neq 0 \\ 0 & \text{otherwise} \end{Bmatrix} = \alpha\,\delta_1(\lambda),$$

where $\delta_1 \in \mathcal{D}_\lambda^{(k)}$ since $\alpha - 1 \leq k$. Thus, the derivative of a function in $\mathcal{D}_\lambda^{(k)}$ w.r.t. λ is also spanned by $\mathcal{D}_\lambda^{(k)}$.

A function $f(\mathbf{q}) \in \operatorname{span}^*\{\Upsilon^{(k)}\}$ is composed of a linear combination of functions $\upsilon_i(\mathbf{q}) \in \Upsilon^{(k)}$. These functions $\upsilon_i(\mathbf{q})$ are themselves a product of a single function $\sigma(q_i)$ in either $\mathcal{Z}_\lambda^{(k)}$ or $\mathcal{D}_\lambda^{(k)}$, and otherwise constant in q_i. Since $\sigma(q_i)$ will remain in the span of its original set under differentiation, then the function $\upsilon_i(\mathbf{q})$ will also stay within the span of $\Upsilon^{(k)}$ under differentiation by an element of $\mathbf{q}$. Since differentiation is linear, this property equally extends to the set of all functions $\mathbf{f}(\mathbf{q})$ in the entrywise span of $\Upsilon^{(k)}$. □

We now demonstrate the relationship between the set of geometric multiplier functions $\Upsilon^{(k)}$ of a manipulator and its kinematic functions, in the following two lemmas.

Lemma 1. *The forward kinematic mappings* ${}^i\mathbf{T}_j(\mathbf{q})$ *between any two frames* j *and* i *of a serial manipulator are entrywise spanned by its* ***geometric multiplier functions of order 1****,* $\Upsilon^{(1)}$.

Proof. Define the variables $(\tilde{a}_i, \tilde{d}_i, \tilde{\alpha}_i, \tilde{\theta}_i)$ as the DH parameters for link lengths, link offsets, link twists and joint angles, respectively, for $i = 1, \ldots, n$. This notation is as defined by Spong and Vidyasagar [27], but we add the tilde to each DH variable to avoid confusion with similarly named variables in this paper. The coordinate change between the robot's $(i-1)$th and ith frame can then be described by

$${}^{i-1}\mathbf{T}_i(q_i) = \begin{bmatrix} \cos\tilde{\theta}_i & -\sin\tilde{\theta}_i\cos\tilde{\alpha}_i & \sin\tilde{\theta}_i\sin\tilde{\alpha}_i & \tilde{a}_i\cos\tilde{\theta}_i \\ \sin\tilde{\theta}_i & \cos\tilde{\theta}_i\cos\tilde{\alpha}_i & -\cos\tilde{\theta}_i\sin\tilde{\alpha}_i & \tilde{a}_i\sin\tilde{\theta}_i \\ 0 & \sin\tilde{\alpha}_i & \cos\tilde{\alpha}_i & \tilde{d}_i \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where, for a revolute joint robot, we have $q_i = \tilde{\theta}_i$ and for a prismatic robot, $q_i = \tilde{d}_i$. By inspection, ${}^{i-1}\mathbf{T}_i(q_i)$ is entrywise spanned by the set $\mathcal{Z}^{(1)}_{q_i}$ if joint i is revolute, and by $\mathcal{D}^{(1)}_{q_i}$ if joint i is prismatic. Writing out the inverse transformation matrix ${}^{i}\mathbf{T}_{i-1}(q_i)$ shows that it is also entrywise spanned by these same sets. Since any transformation between any two frames will be composed of a series of frame transforms,

$$
{}^{i}\mathbf{T}_j(\mathbf{q}) = {}^{i}\mathbf{T}_{i+1}(q_{i+1})\ {}^{i+1}\mathbf{T}_{i+2}(q_{i+2}) \cdots\ {}^{j-1}\mathbf{T}_j(q_j)
$$

where each individual transform is a function of a single joint variable, it follows by definition that the elements of ${}^{i}\mathbf{T}_j(\mathbf{q})$ will be linear combinations of elements of the set $\Upsilon^{(1)}$. □

Lemma 2. *The geometric Jacobian function ${}^{i}\mathbf{J}_j(\mathbf{q})$, to any point $\mathbf{p}$ in frame j relative to a frame i, is entrywise spanned by a serial manipulator's set of **geometric multiplier functions of order 1**, $\Upsilon^{(1)}$.*

Proof. The geometric Jacobian between links is composed of the vertical concatenation of the linear velocity Jacobian $\mathbf{J_v}$ and angular velocity Jacobian $\mathbf{J}_\omega$, where

$$
\mathbf{J_v} = \begin{bmatrix} \dfrac{\partial\, {}^{i}\mathbf{p}_j}{\partial q_i} & \dfrac{\partial\, {}^{i}\mathbf{p}_j}{\partial q_{i+1}} & \cdots & \dfrac{\partial\, {}^{i}\mathbf{p}_j}{\partial q_j} \end{bmatrix}, \qquad \mathbf{J}_\omega = \begin{bmatrix} {}^{i}\mathbf{z}_i & {}^{i}\mathbf{z}_{i+1} & \cdots & {}^{i}\mathbf{z}_{j-1} \end{bmatrix},
$$

and where the vector ${}^{i}\mathbf{z}_j$ is a unit vector of the z-axis of the i^{th} frame relative to the j^{th} frame [27]. The coordinates of ${}^{i}\mathbf{p}_j$ will be in the entrywise span of $\Upsilon^{(1)}$, by Lemma 1. Therefore, by Proposition 2, the partial derivatives of ${}^{i}\mathbf{p}_j$ with respect to each joint variable, as in the definition of $\mathbf{J_v}$, will also be in $\text{span}^*\{\Upsilon^{(1)}\}$. Moreover, since ${}^{i}\mathbf{z}_j$ can be obtained from the first 3 elements of the 3rd column of ${}^{i}\mathbf{T}_j$, it must also be in $\text{span}^*\{\Upsilon^{(1)}\}$ by Lemma 1 [27]. Thus, all the elements of $\mathbf{J_v}$, $\mathbf{J}_\omega$, and by definition, the full Jacobian matrix $\mathbf{J}(\mathbf{q})$, will be in the entrywise span of $\Upsilon^{(1)}$. □

The elements of the transformation matrices of a manipulator, as well as the Jacobian matrices of a manipulator, can therefore be represented as some linear combination of the finite set of functions in $\Upsilon^{(1)}$. We now define two additional sets $\mathcal{A}$ and $\mathcal{Y}$, relating to the accelerations of the manipulator. These definitions will link the geometric multiplier functions to the manipulator dynamics $\Upsilon^{(k)}$, which is the goal of this derivation.

Definition 5. *The sets of functions*

$$
\begin{aligned}
\mathcal{A}_{\ddot{\mathbf{q}}} &= \{\, a : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \ \mid\ a(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \ddot{q}_i, \quad i = 1, \ldots, n \,\}, \\
\mathcal{A}_{\dot{\mathbf{q}}} &= \{\, a : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \ \mid\ a(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \dot{q}_i \dot{q}_j, \ i = 1, \ldots, n, \ j = 1, \ldots, n \,\}, \\
\mathcal{A}_{\mathbf{q}} &= \{\, a : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \ \mid\ a(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) = g \,\},
\end{aligned}
$$

*we define respectively as the **angular**, **velocity** and **gravity acceleration functions** of the manipulator. Furthermore, we define the functions in the union of these sets, $\mathcal{A} = \mathcal{A}_{\ddot{\mathbf{q}}} \cup \mathcal{A}_{\dot{\mathbf{q}}} \cup \mathcal{A}_{\mathbf{q}}$, to be the **acceleration functions** of the manipulator.*

The size of these sets will be relevant in calculations later in this paper. These can be computed as

$$
|\mathcal{A}_{\ddot{\mathbf{q}}}| = n, \qquad |\mathcal{A}_{\dot{\mathbf{q}}}| = \binom{2}{n} + n = \frac{n(n+1)}{2}, \qquad |\mathcal{A}_{\mathbf{q}}| = 1, \qquad |\mathcal{A}| = \frac{(n+1)(n+2)}{2}. \tag{13}
$$

Definition 6. *The **projected acceleration functions of order k** of the manipulator, denoted by $\mathcal{Y}^{(k)}$, is the set of functions obtained through the product of a function from $\mathcal{A}$ and a function from $\Upsilon^{(k)}$, or*

$$
\mathcal{Y}^{(k)} = \left\{ y : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \ \middle|\ y(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = a(\dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \upsilon(\mathbf{q}),\ a(\dot{\mathbf{q}}, \ddot{\mathbf{q}}) \in \mathcal{A},\ \upsilon(\mathbf{q}) \in \Upsilon^{(k)} \right\}. \tag{14}
$$

*We also define the subsets of $\mathcal{Y}^{(k)}$ obtained by the multiplication of an acceleration function from $\mathcal{A}_{\ddot{\mathbf{q}}}$, $\mathcal{A}_{\dot{\mathbf{q}}}$, or $\mathcal{A}_{\mathbf{q}}$, respectively, as the **projected angular**, **velocity**, and **gravity acceleration functions of order k**, and denote them respectively as $\mathcal{Y}^{(k)}_{\ddot{\mathbf{q}}}$, $\mathcal{Y}^{(k)}_{\dot{\mathbf{q}}}$, and $\mathcal{Y}^{(k)}_{\mathbf{q}}$.*

The size of the set of projected acceleration functions is given by the product of the sizes of $\mathcal{A}$ and $\Upsilon^{(k)}$,

$$
|\mathcal{Y}^{(k)}| = |\mathcal{A}| \cdot |\Upsilon^{(k)}| = \frac{(n+1)(n+2)}{2} \cdot \frac{(k+1)^n (k+2)^{n_r}}{2^{n_r}}. \tag{15}
$$

To illustrate these definitions, we show how they apply to the planar manipulator from Figure 1.

Example 2. The sets of acceleration functions of the two-link planar manipulator, as in Definition 5, are

$$\mathcal{A}_{\ddot{\mathbf{q}}} = \{\ddot{q}_1,\ \ \ddot{q}_2\}, \qquad \mathcal{A}_{\dot{\mathbf{q}}} = \{\dot{q}_1^2,\ \ \dot{q}_2^2,\ \ \dot{q}_1\dot{q}_2\}, \qquad \mathcal{A}_{\mathbf{q}} = \{g\}.$$

The sets $\mathcal{Y}^{(1)}$ and $\mathcal{Y}^{(2)}$ are too large to list explicitely. They consist, respectively, of the 54 and 216 possible products of functions from $\Upsilon^{(1)}$ and $\Upsilon^{(2)}$, as per Definition 6, and each of six acceleration functions above. $\triangleleft$

We will now show the relationship between the set of projected acceleration functions $\mathcal{Y}^{(k)}$ and the inverse dynamic model of the manipulator. We do this in three parts, for the gravity, angular acceleration, and centrifugal/Coriolis effects separately. These results will then directly lead to the *Pseudo-Symbolic Representation Theorem*, on which the PSDM algorithm is based.

Lemma 3. *For any serial manipulator, the vector function of joint torques due to gravity, denoted by* $\mathbf{G}(\mathbf{q})$ *as in (1), is entrywise spanned by the set* $\mathcal{Y}_{\mathbf{q}}^{(1)}$ *of* ***projected gravity acceleration functions of order 1****.*

Proof. The gravity vector can be determined from the potential energy function $U(\mathbf{q})$ as

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} \frac{\partial U(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial U(\mathbf{q})}{\partial q_n} \end{bmatrix}^\mathsf{T}, \qquad U(\mathbf{q}) = \mathbf{g}^\mathsf{T}\,\mathbf{r}(\mathbf{q})\,\mathbf{m}, \tag{16}$$

where $\mathbf{r}(\mathbf{q})$ is a matrix of the center of gravity locations in the base frame $(3 \times n)$, $\mathbf{g}$ is the gravity acceleration vector (3×1), and $\mathbf{m}$ is a column vector of link masses $n \times 1$ [27]. Each column of $\mathbf{r}(\mathbf{q})$ is a point transformed linearly by ${}^0\mathbf{T}_i$, $i = 1, \ldots, n$, and therefore by Lemma 1 is entrywise spanned by $\Upsilon^{(1)}$. The potential energy function $U(\mathbf{q})$ is then composed of a linear combination of terms from $\mathbf{r}(\mathbf{q})$, scaled by $g \in \mathcal{A}_{\mathbf{q}}$, and, by definition, is therefore in the entrywise span of $\mathcal{Y}_{\mathbf{q}}^{(1)}$. The desired result follows by Proposition 2. $\square$

Lemma 4. *The vector function* $\boldsymbol{\tau}_{\ddot{\mathbf{q}}}$ *of joint torque due to joint accelerations,* $\boldsymbol{\tau}_{\ddot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{D}(\mathbf{q})\,\ddot{\mathbf{q}}$*, is entrywise spanned by* $\mathcal{Y}_{\ddot{\mathbf{q}}}^{(2)}$*, the* ***projected angular acceleration functions of order 2*** *of the manipulator.*

Proof. For an n-link serial manipulator, the mass matrix $\mathbf{D}(\mathbf{q})$ can be determined from

$$\mathbf{D}(\mathbf{q}) = \sum_{i=1}^{n} \Big[m_i \mathbf{J}_{\mathbf{v}i}^\mathsf{T} \mathbf{J}_{\mathbf{v}i} + \mathbf{J}_{\omega i}^\mathsf{T}\, {}^0\mathbf{R}_i\, \mathbf{I}_i\, {}^0\mathbf{R}_i^\mathsf{T} \mathbf{J}_{\omega i} \Big], \tag{17}$$

where ${}^0\mathbf{R}_i$ is a rotation matrix from frame 0 to i, $\mathbf{J}_{\mathbf{v}i}$ and $\mathbf{J}_{\omega i}$ are the linear and rotation geometric Jacobian matrices of the i^{th} frame (with respect to frame 0), and $\mathbf{I}_i$ and m_i are the inertia matrix and mass of the i^{th} link, respectively [27]. By Lemma 1, the expressions $\mathbf{J}_{\mathbf{v}i}$, and $\mathbf{R}_i^\mathsf{T}\mathbf{J}_{\omega i} = {}^i\mathbf{J}_{\omega i}$ are in the entrywise span of $\Upsilon^{(1)}$. $\mathbf{D}(\mathbf{q})$ is then the sum of two products of matrix functions in $\text{span}^*\{\Upsilon^{(1)}\}$, so by Proposition 1, $\mathbf{D}(\mathbf{q}) \in \text{span}^*\{\Upsilon^{(2)}\}$. The result follows by the definition of $\mathcal{Y}_{\ddot{\mathbf{q}}}^{(2)}$ in (14) since $\boldsymbol{\tau}_{\ddot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{D}(\mathbf{q})\,\ddot{\mathbf{q}}$. $\square$

Lemma 5. *For any serial manipulator, the vector function* $\boldsymbol{\tau}_{\dot{\mathbf{q}}}$ *of joint torques due to centrifugal and Coriolis accelerations,* $\boldsymbol{\tau}_{\dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$*, is entrywise spanned by* $\mathcal{Y}_{\dot{\mathbf{q}}}^{(2)}$*, the* ***projected velocity acceleration functions of order 2*** *of the manipulator.*

Proof. It is well documented and can be derived directly from the Euler-Lagrange formulation of robot dynamics [27] that the torque in joint k due to centrifugal and Coriolis terms, $\boldsymbol{\tau}_{\dot{\mathbf{q}},k}$, can be rewritten as

$$\tau_{\dot{\mathbf{q}},k} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ijk}(\mathbf{q})\, \dot{q}_i\, \dot{q}_j, \qquad k = 1, \ldots, n, \tag{18}$$

where $c_{ijk}(\mathbf{q})$ are the Christoffel symbols of the first kind, and are obtained from a linear combination of derivatives of the elements $d_{ij}(\mathbf{q})$ of $\mathbf{D}(\mathbf{q})$ as

$$c_{ijk}(\mathbf{q}) = \frac{1}{2} \left(\frac{\partial}{\partial q_i} d_{kj}(\mathbf{q}) + \frac{\partial}{\partial q_j} d_{ki}(\mathbf{q}) + \frac{\partial}{\partial q_k} d_{ij}(\mathbf{q}) \right). \tag{19}$$

Since by Lemma 4, $\mathbf{D}(\mathbf{q})$ is entrywise spanned by $\Upsilon^{(2)}$, it follows from Proposition 2 that functions $c_{ijk}(\mathbf{q})$, consisting of a sum of derivatives of $\mathbf{D}(\mathbf{q})$, will also be entrywise spanned by $\Upsilon^{(2)}$. By inspection of (18), we can therefore conclude that $\boldsymbol{\tau}_{\dot{\mathbf{q}}}$ must also be in the entrywise span of $\mathcal{Y}^{(2)}$. $\square$

These Lemmas lead us to the main result of this section, the *Pseudo-Symbolic Representation Theorem.*

Proposition 3. *(Pseudo-Symbolic Representation Theorem) The inverse dynamics function of any serial kinematic manipulator,* $\boldsymbol{\tau} = \mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q})$*, lies within the entrywise span of* $\mathcal{Y}^{(2)}$*, its* ***projected acceleration functions of order 2****.*

Proposition 3 follows directly from Lemmas 3 to 5. However, the result is important. It states that the inverse dynamics function, for any joint of an arbitrary serial robotic manipulator, can be exactly represented as some linear combination of functions in $\mathcal{Y}^{(2)}$. Therefore, it follows that we must be able to compute the manipulator dynamics in the same form as the regression equation (3), using the functions in $\mathcal{Y}^{(2)}$, as

$$\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})^{\mathsf{T}} = \begin{bmatrix} y_1 & \cdots & y_t \end{bmatrix} \begin{bmatrix} \theta_{1,1} & \cdots & \theta_{1,n} \\ \vdots & \ddots & \vdots \\ \theta_{t,1} & \cdots & \theta_{t,n} \end{bmatrix} = \mathbf{y}_t(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\boldsymbol{\Theta}_t, \tag{20}$$

where $\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ is an n-element column vector of joint torques and the functions y_j are well-defined functions from a finite set $\mathcal{Y}^{(2)}$. The size t of this set is calculated from (15) as

$$t \triangleq \left|\mathcal{Y}^{(2)}\right| = 6^{n_r}\, 3^{n_p}\, \frac{(n+1)(n+2)}{2}. \tag{21}$$

Moreover, the linear map between $\mathbf{y}_t(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ and the joint torques $\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ are the identifiable inertial parameters. Therefore, Proposition 3 gives us a generalized, albeit non-minimal, expression for the inverse dynamics of a manipulator, in regressor form as in (3).

Example 3. The Pseudo-Symbolic Representation Theorem can be verified by inspection for the planar manipulator from Figure 1. The full inverse dynamic model of this manipulator can be derived using the Euler-Lagrange equations, [27]. Grouping and collecting terms appropriately, these equations are

$$\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})^{\mathsf{T}} = \begin{bmatrix} \tau_1 & \tau_2 \end{bmatrix} = \mathbf{y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\boldsymbol{\Theta} \tag{22}$$

with the vector function $\mathbf{y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ and matrix $\boldsymbol{\Theta}$ defined as

$$\begin{aligned}
\mathbf{y}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}}) &= \begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} \end{bmatrix} \\
&= \begin{bmatrix} \upsilon_1\ddot{q}_1 & \upsilon_3\ddot{q}_1 & \upsilon_1\ddot{q}_2 & \upsilon_3\ddot{q}_2 & \upsilon_4\dot{q}_1^2 & \upsilon_4\dot{q}_2^2 & \upsilon_4\dot{q}_1\dot{q}_2 & \upsilon_6 g & \upsilon_2 g & \upsilon_5 g \end{bmatrix} \\
\boldsymbol{\Theta}^{\mathsf{T}} &= \begin{bmatrix} \theta_{1,1} & \theta_{2,1} & \theta_{3,1} & \theta_{4,1} & 0 & \theta_{6,1} & \theta_{7,1} & \theta_{8,1} & \theta_{9,1} & \theta_{10,1} \\ \theta_{1,2} & \theta_{2,2} & \theta_{3,2} & 0 & \theta_{5,2} & 0 & 0 & \theta_{8,2} & 0 & \theta_{10,2} \end{bmatrix}
\end{aligned}$$

The functions υ_j correspond to functions in the set $\Upsilon^{(2)}$ and are defined as

$$\upsilon_1 = 1, \qquad \upsilon_2 = c_1, \qquad \upsilon_3 = c_2, \qquad \upsilon_4 = s_2, \qquad \upsilon_5 = c_1 c_2, \qquad \upsilon_6 = s_1 s_2,$$

and the inertial parameters are

$$\begin{aligned}
\theta_{1,1} &= I_{zz,1} + I_{zz,2} + m_1\left(a_1 + r_{x,1}\right)^2 + m_2\left(a_2 + r_{x,2}\right)^2 + {a_1}^2 m_2, \\
\theta_{2,1} &= -\theta_{7,1} = 2\,\theta_{4,1} = 2\,\theta_{2,2} = -2\,\theta_{6,1} = 2\,\theta_{5,2} = 2m_2 a_1\left(a_2 + r_{x,2}\right), \\
\theta_{3,1} &= \theta_{1,2} = \theta_{3,2} = m_2\left(a_2 + r_{x,2}\right)^2 + I_{zz,2}, \\
\theta_{8,1} &= \theta_{8,2} = -\theta_{10,1} = -\theta_{10,2} = -m_2\left(a_2 + r_{x,2}\right), \\
\theta_{9,1} &= m_2 a_1 + m_1\left(a_1 + r_{x,1}\right).
\end{aligned}$$

By inspection, in the given arrangement, $\boldsymbol{\tau}$ is composed of a linear combination of functions y_j, all of which are elements of the set $\mathcal{Y}^{(2)}$, as identified previously. Moreover, the parameter matrix $\boldsymbol{\Theta}$ is solely a function of the inertial and kinematic parameters of the manipulator, so these equations are in regressor form. $\lhd$

2.2. Reduction to Minimal Form

The *Pseudo-Symbolic Representation Theorem* identifies t functions which are guaranteed to span the inverse dynamics function with some inertial parameter matrix $\boldsymbol{\Theta}_t$. To use this set of functions with PSDM, we want to reduce (20) to a minimal form. This reduction is done in two steps.

First, although t functions are identified to entrywise span $\boldsymbol{\tau}$, the majority of these functions are not required to form a minimal basis. In the linear set of equations in (20), some functions will have corresponding rows in the parameter matrix $\boldsymbol{\Theta}_t$ that are identically zero, and as such, have zero contribution to $\boldsymbol{\tau}$. These "zero" terms should be eliminated from the model in order to obtain the set of p functions in the set $\mathcal{Y}_p \subseteq \mathcal{Y}^{(2)}$, which we call the *base regressor functions*, such that

$$\boldsymbol{\tau}^{\mathsf{T}} = \begin{bmatrix}\tau_1 & \cdots & \tau_n\end{bmatrix} = \begin{bmatrix}y_1 & \cdots & y_p\end{bmatrix} \begin{bmatrix}\theta_{1,1} & \cdots & \theta_{1,n} \\ \vdots & \ddots & \vdots \\ \theta_{p,1} & \cdots & \theta_{p,n}\end{bmatrix} = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \boldsymbol{\Theta}_p, \tag{23}$$

which is identical to (20) except that $\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ now minimally contains $p \ll t$ columns and the regression matrix $\boldsymbol{\Theta}_p$ is similarly reduced in size.

Second, the reduced parameter matrix $\boldsymbol{\Theta}_p \in \mathbb{R}^{p \times n}$, which linearly relates the base regressor terms to each of the joint torques as in (23), can have internal linear dependencies that should be eliminated. This elimination process reduces the parameter matrix $\boldsymbol{\Theta}_p$ to a single ℓ-vector $\boldsymbol{\theta}_b$ of **base inertial parameters**, which are minimally required to calibrate the dynamic model. These two steps are described in the following sections.

2.2.1. Identification of Base Regressor Functions

In this section, we show how we can identify the set $\mathcal{Y}_p$ of p functions, which are minimally required to form a basis for $\boldsymbol{\tau}$, from the much larger set of t functions in the set $\mathcal{Y}^{(2)}$ as identified by Proposition 3. For example, in the formulation given for the planar manipulator from Examples 1 to 3, this process corresponds to reducing the set of $t = 216$ functions in $\mathcal{Y}^{(2)}$ to the smaller set of $p = 10$ functions identified in (22).

This process can be reformulated as a system identification problem. Equation (20) is in a form convenient for linear regression to identify $\boldsymbol{\Theta}_t$, given sampled data from $\boldsymbol{\tau}$. If, in any full row j of the identified $\boldsymbol{\Theta}_t$, all elements are zero, then corresponding function y_j can be eliminated from the dynamic model. Moreover, this sampled data does not need to be experimental. Here we can leverage any numerical dynamic model, such as the recursive Newton-Euler (RNE) method [15], or any other dynamic model (which need not be minimal, efficient, or in regressor form), to generate samples of the joint torques for chosen values of the joint states $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and some arbitrary non-zero set of inertial parameters $\mathbf{X}$. Moreover, since we are able to select the sample points, we can make the system identification problem maximally exciting using synthetic random data. The identification problem can therefore be performed where the only noise in the problem is machine precision error. Methodologies for selecting appropriate sample points are discussed later in this paper, in Section 3.3. With $N \geq t$ joint states $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and a single set of arbitrary inertial parameters $\mathbf{X}$, we compute an $N \times t$ matrix $\mathbf{Y}_t$ and an $n \times N$ torque matrix $\mathbf{T}_t$ where

$$\mathbf{Y}_t = \begin{bmatrix} y_1(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1) & \cdots & y_t(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1) \\ \vdots & \ddots & \vdots \\ y_1(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N) & \cdots & y_t(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N) \end{bmatrix}, \quad \mathbf{T}_t = \begin{bmatrix} \tau_1(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1, \mathbf{X}) & \cdots & \tau_1(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N, \mathbf{X}) \\ \vdots & \ddots & \vdots \\ \tau_n(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1, \mathbf{X}) & \cdots & \tau_n(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N, \mathbf{X}) \end{bmatrix}. \tag{24}$$

The parameter matrix $\boldsymbol{\Theta}_t$ can be computed as the exact (if $N = t$) or least-squares (if $N > t$) solution to the system of linear equations

$$\mathbf{T}_t^{\mathsf{T}} = \mathbf{Y}_t \boldsymbol{\Theta}_t, \tag{25}$$

and can be solved using any appropriate linear system solution algorithm. With $\boldsymbol{\Theta}_t$ known, we can readily identify the functions which are in the minimal set $\mathcal{Y}_p$, since they will have non-zero values in the corresponding rows of $\boldsymbol{\Theta}_t$. A column j of $\mathbf{Y}_t$ has zero correlation to $\mathbf{T}_t$ if

$$\left|\theta_{j,i}\right| < \varepsilon, \quad i = 1, \ldots, n, \tag{26}$$

and where ε is an appropriate numerical tolerance to account for round-off errors. Any column j that has zero correlation to $\mathbf{T}_t$ corresponds to a function $y_j \in \mathcal{Y}^{(2)}$ that is not correlated to $\boldsymbol{\tau}$. As such, it can safely be eliminated from the model without any effect.

After eliminating all appropriate functions as per (26), we are left with the p functions in $\mathcal{Y}_p$, which are minimally necessary to span $\boldsymbol{\tau}$. Grouping the functions in $\mathcal{Y}_p$ in the same manner as (20) gives the reduced vector function $\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, such that the inverse dynamics function can be equivalently written for some $p \times n$ matrix of inertial parameters $\boldsymbol{\Theta}_p$ as stated earlier in (23).

2.2.2. Reduction to Base Inertial Parameters

It is important to note that the size p of the set of *base regressor terms* $\mathcal{Y}_p$ is not the same as the size ℓ of the *base inertial parameters* $\boldsymbol{\theta}_b$. After $\mathbf{Y}_p$ has been found, the corresponding parameters $\boldsymbol{\Theta}_p$ can still have internal linear dependencies w.r.t. to the individual inertial parameters $\mathbf{X}$. For example, in the case of the two-link planar manipulator in Example 3, one may note that of the 15 non-zero elements of $\boldsymbol{\Theta}_p$ in (22), many were simple linear combinations of one another. In fact, it will be shown later in this paper that there are only four parameters that are required to fully calibrate the dynamic model of this robot. The internal linear dependencies must be removed to determine the dynamic model in terms of a base inertial parameter set. To accomplish this reduction, we search for n pairs of *reduction matrices* $\mathbf{P}_i$ and $\mathbf{B}_i$, of size $p \times \ell$ and $\ell \times p$, respectively. If we define $\boldsymbol{\theta}_{p,i}$ as the i^{th} column of $\boldsymbol{\Theta}_p$,

$$\boldsymbol{\Theta}_p = \begin{bmatrix} \boldsymbol{\theta}_{p,1} & \cdots & \boldsymbol{\theta}_{p,n} \end{bmatrix}, \tag{27}$$

then for each joint i, these matrices should map $\boldsymbol{\theta}_{p,i} \in \mathbb{R}^p$ to a set of *base inertial parameters* $\boldsymbol{\theta}_b \in \mathbb{R}^\ell$, as

$$\boldsymbol{\theta}_b = \mathbf{B}_i \boldsymbol{\theta}_{p,i}, \qquad i = 1, \dots, n. \tag{28}$$

Additionally, we require that each of these matrices can be inserted in the regression equation of (23) without modifying the resulting torques, such that it is possible to state that

$$\tau_i = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \boldsymbol{\theta}_{p,i} = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{P}_i\, \mathbf{B}_i\, \boldsymbol{\theta}_{p,i} = \begin{bmatrix} \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{P}_i \end{bmatrix} \boldsymbol{\theta}_b, \qquad i = 1, \dots, n. \tag{29}$$

The matrices $\mathbf{P}_i, \mathbf{B}_i$ are not unique, and the problem is not as trivial as it may appear. Gautier [20] presented a solution to this problem using QR decomposition. In the current work, we use an approach which derives from the work of Gautier, but with some implementational and notational differences. Similar to the previous section, we generate $N \geq p$ random joint states $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. Additionally, we also generate $M \geq \ell$ random inertial parameter matrices $\mathbf{X}_i$. Here, since ℓ is not known beforehand, we can instead use $M \geq 10n$ since ℓ must necessarily be less or equal to the total number of inertial parameters in $\mathbf{X}$,

$$\ell \leq 10n \leq M. \tag{30}$$

From these samples, using the RNE (reverse Newton-Euler) method, or some other dynamic model, we construct n matrices $\mathsf{T}_i \in \mathbb{R}^{N \times M}$, $i = 1, \dots, n$, for each joint, as

$$\mathsf{T}_i = \begin{bmatrix} \tau_i(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1, \mathbf{X}_1) & \cdots & \tau_i(\mathbf{q}_1, \dot{\mathbf{q}}_1, \ddot{\mathbf{q}}_1, \mathbf{X}_M) \\ \vdots & \ddots & \vdots \\ \tau_i(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N, \mathbf{X}_1) & \cdots & \tau_i(\mathbf{q}_N, \dot{\mathbf{q}}_N, \ddot{\mathbf{q}}_N, \mathbf{X}_M) \end{bmatrix}. \tag{31}$$

Additionally, we compute the corresponding $N \times p$ regression matrix $\mathbf{Y}_p$ from the sample joint states. This matrix has the same equation as in (24), but uses the functions from the reduced set $\mathcal{Y}_p$, rather than the full set $\mathcal{Y}^{(2)}$. Assuming sampling is done such that $\mathbf{Y}_p$ is well-conditioned, a $p \times M$ matrix of regression parameters $\boldsymbol{\Psi}_i$, for each joint i, can be determined as the solution to the linear system of equations

$$\mathsf{T}_i = \mathbf{Y}_p \boldsymbol{\Psi}_i, \tag{32}$$

where $\boldsymbol{\Psi}_i$ will contain M columns of regression vectors $\boldsymbol{\theta}_{p,i}$ for the system, each corresponding to a sample of the full inertial parameters, as

$$\boldsymbol{\Psi}_i = \begin{bmatrix} \boldsymbol{\theta}_{p,i}\big|_{\mathbf{X}=\mathbf{X}_1} & \cdots & \boldsymbol{\theta}_{p,i}\big|_{\mathbf{X}=\mathbf{X}_M} \end{bmatrix}. \tag{33}$$

First, let us consider reducing the inertial parameters for each joint i in isolation, since extending the analysis to the full manipulator afterward is not overly complex. For this purpose, let us define $\tilde{\mathbf{P}}_i$ and $\tilde{\mathbf{B}}_i$, of dimension $p \times \ell_i$ and $\ell_i \times p$, respectively, where ℓ_i is the minimum number of regression parameters required to calibrate solely the torques of the robot's i^{th} joint. Because $\boldsymbol{\Psi}_i$ is a horizontal concatenation of possible regression vectors for joint i, sampled across the space of possible inertial parameters $\mathbf{X}$, any linear dependencies between the elements of $\boldsymbol{\theta}_{p,i}$ will necessarily appear as numerical linear dependencies in the rows of $\boldsymbol{\Psi}_i$. Therefore, the number of linearly independent rows of $\boldsymbol{\Psi}_i$ will be exactly equal to the number of linearly independent elements in $\boldsymbol{\theta}_{p,i}$, and so necessarily $\ell_i = \operatorname{rank}(\boldsymbol{\Psi}_i)$. Moreover, the internal linear dependencies of the inertial parameters can be numerically determined directly from $\boldsymbol{\Psi}_i$. A possible set of solutions to (29) is

$$\tilde{\mathbf{B}}_i = \mathbf{A}\, {\boldsymbol{\Psi}'_i}^{\mathsf{T}}, \qquad\qquad \tilde{\mathbf{P}}_i = \tilde{\mathbf{B}}_i^{\dagger}, \tag{34}$$

where $\boldsymbol{\Psi}'_i \in \mathbb{R}^{p \times \ell_i}$ is a matrix formed by any ℓ_i linearly independent columns of $\boldsymbol{\Psi}_i$, $\mathbf{A} \in \mathbb{R}^{\ell_i \times \ell_i}$ is any full rank and invertible matrix, and the superscript $\dagger$ represents the Moore-Penrose pseudoinverse. Note that this solution is not arbitrary; in general, $\tilde{\mathbf{P}}_i\tilde{\mathbf{B}}_i = \tilde{\mathbf{B}}_i^{\dagger}\tilde{\mathbf{B}}_i = \mathbf{I}$, a condition that would satisfy (29), is *only* true if $\tilde{\mathbf{B}}_i$ has linearly independent columns. However, $\tilde{\mathbf{B}}_i$ cannot have linearly independent columns since it is of size $\ell_i \times p$, where $p > \ell_i$. However, with (34), the equality does hold. While $\tilde{\mathbf{P}}_i\tilde{\mathbf{B}}_i = \tilde{\mathbf{B}}_i^{\dagger}\tilde{\mathbf{B}}_i = \mathbf{I}$ is *not* true in general, the matrix $\left[\tilde{\mathbf{P}}_i\tilde{\mathbf{B}}_i\right]$ can always be described as an orthogonal projector mapping from $\mathbb{R}^p$ to the nearest point in the range space of $\tilde{\mathbf{B}}_i^{\mathsf{T}}$ [28, p. 257]. If $\tilde{\mathbf{B}}_i$ has linearly independent columns, this is equivalent to saying that $\left[\tilde{\mathbf{P}}_i\tilde{\mathbf{B}}_i\right] = \mathbf{I}$. Otherwise, $\left[\tilde{\mathbf{P}}_i\tilde{\mathbf{B}}_i\right] \neq \mathbf{I}$, however, when applied to any vector *already* in the range space of $\tilde{\mathbf{B}}_i^{\mathsf{T}}$, it will still act as an identity mapping. Since the definition of $\tilde{\mathbf{B}}_i$ in (34) necessarily ensures that any regression vector $\boldsymbol{\theta}_{p,i}$ is in the range space of $\tilde{\mathbf{B}}_i^{\mathsf{T}}$, we can say that

$$\left[\tilde{\mathbf{P}}_i\,\tilde{\mathbf{B}}_i\right]\boldsymbol{\theta}_{p,i} = \boldsymbol{\theta}_{p,i}, \tag{35}$$

and so (29) is satisfied as

$$\left[\,\mathbf{y}_p(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\tilde{\mathbf{P}}_i\,\right]\boldsymbol{\theta}_b = \mathbf{y}_p(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\tilde{\mathbf{P}}_i\,\tilde{\mathbf{B}}_i\,\boldsymbol{\theta}_{p,i} = \mathbf{y}_p(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})\,\boldsymbol{\theta}_{p,i} = \tau_i, \qquad i = 1,\ldots,n. \tag{36}$$

For example, one could let $\mathbf{A} = \mathbf{I}_{\ell_i\times\ell_i}$ and $\boldsymbol{\Psi}_i'$ be the first ℓ_i columns of $\boldsymbol{\Psi}_i$. In theory, this would give the desired result, but has two issues. The first is numerical – rather than select the first ℓ_i linearly independent columns of $\boldsymbol{\Psi}_i$, it is better to select them such that the resulting matrix is as well-conditioned as possible. The second regards efficiency and elegance – with this solution, the resulting reduction matrices are dense and contain no easily visible structure. However, we know that the inertial parameters should be composed of simple linear combinations of one another. Thus, we should be able to find a set of reduction matrices which leverage this fact to more clearly illustrate the relationship between the parameters, while also being sparse – composed primarily of zeros, ones, and simple rational functions of the kinematic manipulator parameters. Such a form would give more insight into the dynamic model itself, and also would allow for computational optimizations. If the reduction matrices are sparse, many of the multiplications by zero can be removed from the model. Consider instead letting $\mathbf{A} = \tilde{\boldsymbol{\Psi}}_i^{-\mathsf{T}}$, such that

$$\tilde{\mathbf{B}}_i = \tilde{\boldsymbol{\Psi}}_i^{-\mathsf{T}}\,\boldsymbol{\Psi}_i'^{\mathsf{T}}, \tag{37}$$

where $\tilde{\boldsymbol{\Psi}}_i \in \mathbb{R}^{\ell_i\times\ell_i}$ is the first ℓ_i linearly independent *rows* of $\boldsymbol{\Psi}_i'$. Then, $\mathbf{A}$ is by definition full rank and invertible, and we obtain a particularly elegant solution, which readers may recognize as equivalent to letting $\tilde{\mathbf{B}}_i$ be the *row-reduced echelon form* of $\boldsymbol{\Psi}_i^{\mathsf{T}}$, trimmed down to dimension $\ell_i \times p$. With this solution, the first ℓ_i columns of $\tilde{\mathbf{B}}_i$ become the identity matrix, and the remaining columns of $\tilde{\mathbf{B}}_i$ indicate the representation of corresponding rows of $\boldsymbol{\Psi}_i$ with respect to the first ℓ_i linearly independent rows used in $\tilde{\boldsymbol{\Psi}}_i$. This mapping therefore preserves the first ℓ_i linearly independent parameters in $\boldsymbol{\Psi}_i$, and eliminates the rest by representing them as linear combinations of the rows of $\tilde{\boldsymbol{\Psi}}_i$. The reduction matrices $\tilde{\mathbf{P}}_i$ and $\tilde{\mathbf{B}}_i$ are then mostly composed of zeros, ones and small rational functions of the robot's kinematic parameters.

The remaining problem is how one can determine the quantities $\tilde{\boldsymbol{\Psi}}_i$ and $\boldsymbol{\Psi}_i'$ in a numerically robust way. One method is by using QR decomposition. The matrix $\boldsymbol{\Psi}_i^{\mathsf{T}}$ can be decomposed into [29]

$$\boldsymbol{\Psi}_i^{\mathsf{T}} = \mathbf{Q}\,\mathbf{R}\,\boldsymbol{\Pi}^{\mathsf{T}} = \begin{bmatrix}\mathbf{Q}_1 & * \\ \mathbf{Q}_2 & *\end{bmatrix}\begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2 \\ \mathbf{0} & \mathbf{R}_3\end{bmatrix}\boldsymbol{\Pi}^{\mathsf{T}}, \tag{38}$$

where

- $\mathbf{Q} \in \mathbb{R}^{M\times M}$ is an orthogonal matrix and has sub-matrices $\mathbf{Q}_1 \in \mathbb{R}^{\ell_i\times\ell_i}$ and $\mathbf{Q}_2 \in \mathbb{R}^{(M-\ell_i)\times\ell_i}$;
- $\mathbf{R} \in \mathbb{R}^{M\times p}$ is a well-conditioned, upper triangular matrix with diagonal elements whose magnitude is non-increasing. Additionally, $\mathbf{R}$ can be divided into sub-matrices $\mathbf{R}_1 \in \mathbb{R}^{\ell_i\times\ell_i}$, $\mathbf{R}_2 \in \mathbb{R}^{\ell_i\times(p-\ell_i)}$ and $\mathbf{R}_3 \in \mathbb{R}^{(M-\ell_i)\times(p-\ell_i)}$, such that $\|\mathbf{R}_3\|_2$ is small and $\mathbf{R}_3 = \mathbf{0}$ within some tolerance ε;
- $\boldsymbol{\Pi}$ is a permutation matrix, such that $\boldsymbol{\Pi}^{-1} = \boldsymbol{\Pi}^{\mathsf{T}}$.

This decomposition is the so-called *rank-revealing QR factorization*, and conveniently allows the rank ℓ_i to be estimated with a tolerance $\varepsilon_{\mathrm{QR}}$ as

$$\varepsilon_{\mathrm{QR}} = |R_{11}|\,p\,\varepsilon_0, \tag{39}$$

where $|R_{11}|$ is the magnitude of the first diagonal element of $\mathbf{R}$, ε_0 is the machine precision, such that ℓ_i can be calculated as the number of diagonal elements of $\mathbf{R}$ which are greater than $\varepsilon_{\mathrm{QR}}$ [30]. After some rearrangements, the necessary matrices can be computed with good numerical stability as

$$\mathbf{\Psi}_i'^{\,\mathsf{T}} = \mathbf{Q}_1 \begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2\end{bmatrix} \mathbf{\Pi}^{\mathsf{T}}, \qquad \tilde{\mathbf{\Psi}}_i^{\mathsf{T}} = \mathbf{Q}_1 \mathbf{R}_1 \mathbf{\Pi}_1, \tag{40}$$

and $\mathbf{\Pi}_1 \in \mathbb{R}^{\ell_i \times \ell_i}$ is a matrix formed by taking the first ℓ_i rows of $\mathbf{\Pi}$, then removing all but the non-zero columns. Substitution of (40) into (37) yields the final, simplified equation,

$$\begin{aligned}\tilde{\mathbf{B}}_i = \tilde{\mathbf{\Psi}}_i^{-\mathsf{T}} \mathbf{\Psi}_i'^{\,\mathsf{T}} &= \left(\mathbf{Q}_1 \mathbf{R}_1 \mathbf{\Pi}_1\right)^{-1} \mathbf{Q}_1 \begin{bmatrix}\mathbf{R}_1 & \mathbf{R}_2\end{bmatrix} \mathbf{\Pi}^{\mathsf{T}} \\ &= \mathbf{\Pi}_1^{\mathsf{T}} \begin{bmatrix}\mathbf{I} & \mathbf{R}_1^{-1}\mathbf{R}_2\end{bmatrix} \mathbf{\Pi}^{\mathsf{T}}.\end{aligned} \tag{41}$$

Until now, we have been considering reducing the inverse dynamic equations for joint i in isolation. However, we would like to consider a single set of base inertial parameters for the entire manipulator, not n separate sets, specific to each joint. Fortunately, however, the solution to this problem is straightforward. Consider the matrix $\mathbf{\Psi}_*$ formed by horizontally concatenating the joint-specific $\mathbf{\Psi}_i$ as

$$\mathbf{\Psi}_* = \begin{bmatrix}\mathbf{\Psi}_1 & \cdots & \mathbf{\Psi}_n\end{bmatrix}. \tag{42}$$

If we instead perform the analysis above on the matrix $\mathbf{\Psi}_*$, rather than each individual matrix $\mathbf{\Psi}_i$, the inertial effects of all the joints will be considered simultaneously. Accordingly, the rank of this concatenated matrix must be equal to the minimal number of parameters for the entire manipulator dynamic model,

$$\ell = \operatorname{rank}(\mathbf{\Psi}_*). \tag{43}$$

Moreover, if we substitute $\mathbf{\Psi}_*$ in place of $\mathbf{\Psi}_i$ in equations (37), (38) and (41), the numerical analysis will return stacked reduction matrices $\mathbf{B}_* \in \mathbb{R}^{\ell \times np}$ and $\mathbf{P}_* \in \mathbb{R}^{np \times \ell}$, which resolve both the internal dependencies in the inertial parameters of each joint, as well as the dependencies in parameters between joints, as

$$\mathbf{B}_* = \begin{bmatrix}\mathbf{B}_1 & \cdots & \mathbf{B}_n\end{bmatrix}, \qquad \mathbf{P}_* = \begin{bmatrix}\mathbf{P}_1 \\ \vdots \\ \mathbf{P}_n\end{bmatrix} = \mathbf{B}_*^{\dagger}. \tag{44}$$

The reduction matrices $\mathbf{B}_i$ and $\mathbf{P}_i$, can then recovered by slicing the stacked matrices above appropriately. With $\mathbf{P}_i$ and $\mathbf{B}_i$ known, the regression equation from (23) can be rewritten in terms of a single ℓ-vector of base inertial parameters $\boldsymbol{\theta}_b$, similarly to the regressor equation in (3), as

$$\tau_i(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{P}_i\, \mathbf{B}_i\, \boldsymbol{\theta}_{p,i} = \begin{bmatrix}\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{P}_i\end{bmatrix} \boldsymbol{\theta}_b, \qquad i = 1, \ldots, n. \tag{45}$$

Equation (45) can now be used as the inverse dynamic model for the robot. The base inertial parameter ℓ-vector $\boldsymbol{\theta}_b$ can be estimated from experimental measurements, or from full knowledge of the inertial parameters $\mathbf{X}$, using RNE sampling, as before. Additionally, if the inertial parameters $\boldsymbol{\theta}_b$ are constant for the application, $\mathbf{P}_i$ and $\boldsymbol{\theta}_b$ can be pre-multiplied and concatenated into a $p \times n$ matrix $\mathbf{\Phi}_b$, as

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{\Phi}_b, \qquad \mathbf{\Phi}_b \triangleq \begin{bmatrix}\mathbf{P}_1 \boldsymbol{\theta}_b & \cdots & \mathbf{P}_n \boldsymbol{\theta}_b\end{bmatrix}. \tag{46}$$

3. Implementational Notes

This section will discuss several practical considerations and simplifications for the implementation of PSDM in code. Section 3.1 discusses simplifications that are possible for the algorithm. Section 3.2 presents a numerical method for encoding, storing and computing the many functions $y_j \in \mathcal{Y}_p$. Section 3.3 discusses how to appropriately choose joint and inertial parameter samples for the derivation, and Section 3.4 examines methods for solving the linear systems in (25) and (32). Section 3.5 discusses how additional dynamic effects, such as drive inertias or joint friction, can be incorporated into the PSDM method. Finally, Section 3.6 gives a full summary of the steps of the PSDM method and demonstrates it in a complete example.

3.1. Algorithm Simplifications

In PSDM, the bulk of the computation required is in (25), where the system of equations $\mathbf{T}_t^{\mathsf{T}} = \mathbf{Y}_t \mathbf{\Theta}_t$ must be solved to identify the base regressor terms $\mathcal{Y}_p$. This step involves solving a $t \times t$ linear system, where t is as defined in (21) and can be quite large for more complex robots. Every subsequent step involves only small computations, relatively speaking, since the size of the base regressor functions p is much smaller than t. It is therefore desirable to reduce the complexity of this step as much as possible. In this section, we discuss two techniques for reducing the complexity of this computation. Section 3.1.1 shows how the linear system in (25) can be partitioned into smaller problems, and Section 3.1.2 shows how a number of simple pre-filters can be applied to the set $\mathcal{Y}^{(2)}$ to further reduce the size of the linear system to be solved.

3.1.1. *Partitioning of Linear Systems*

The accelerations of the robot are divided into n *angular* accelerations, $\binom{n}{2}$ *Coriolis* accelerations, n *centrifugal* accelerations and a single *gravity* acceleration. Each function $y_j \in \mathcal{Y}^{(2)}$ is a function of only one these accelerations. Furthermore, by carefully selecting the points at which $\boldsymbol{\tau}$ is sampled, we can isolate the torques due to each type of acceleration as

$$\begin{aligned} \boldsymbol{\tau}_{\text{grav}} &= \boldsymbol{\tau}(\mathbf{0},\, \mathbf{0},\, \mathbf{q}), & \boldsymbol{\tau}_{\text{ang},i} &= \boldsymbol{\tau}(\ddot{\mathbf{q}}_{\{i\}},\, \mathbf{0},\, \mathbf{q}) \;\; - \boldsymbol{\tau}_{\text{grav}}, \\ \boldsymbol{\tau}_{\text{cent},i} &= \boldsymbol{\tau}(\mathbf{0},\, \dot{\mathbf{q}}_{\{i\}},\, \mathbf{q}) - \boldsymbol{\tau}_{\text{grav}}, & \boldsymbol{\tau}_{\text{cor},ij} &= \boldsymbol{\tau}(\mathbf{0},\, \dot{\mathbf{q}}_{\{i,j\}},\, \mathbf{q}) - \boldsymbol{\tau}_{\text{grav}} - \boldsymbol{\tau}_{\text{cent},i} - \boldsymbol{\tau}_{\text{cent},j}, \end{aligned} \tag{47}$$

where we use the symbols $\ddot{\mathbf{q}}_{\{k\}}$, $\dot{\mathbf{q}}_{\{k\}}$, and $\dot{\mathbf{q}}_{\{j,k\}}$ as an n-vector of zeros, except in the k^{th} (or j^{th} and k^{th}) positions. Because the contribution of each acceleration to the dynamic model is independent of the others, the system of linear equations in (25) can be divided into multiple smaller systems, one for each acceleration, rather than solving the system in a single step. We isolate the torque samples to an acceleration $a \in \mathcal{A}$, then consider only the columns of $\mathbf{Y}_t$ which involve the isolated acceleration. In this way, we can take the $t \times t$ linear system, where $t = \left|\mathcal{Y}^{(2)}\right| = 6^{n_r}\, 3^{n_p}\, (n+1)(n+2)/2$, and reduce it into $|\mathcal{A}| = (n+1)(n+2)/2$ smaller linear systems of size $6^{n_r}\, 3^{n_p} = \left|\Upsilon^{(2)}\right|$. After each sub-system has been solved, and the non-essential functions in each one eliminated, the overall system can then be reconstructed by appropriately concatenating the results. Since the complexity of solving a linear system scales exponentially, solving these problems separately is significantly less computationally demanding than solving them together.

3.1.2. *Pre-Filtering of Search Functions*

While the functions of $\mathcal{Y}^{(2)}$ are guaranteed to form a basis for the elements of $\boldsymbol{\tau}$, not all the functions in this set are required. In fact, there are some additional simplifications that can be immediately applied to eliminate many functions from this set, before solving the linear system in (25). These filters further reduced the size of the linear systems to be solved. Note that these pre-filters are not necessary to leverage PSDM. However, their use can greatly increase the speed of the PSDM derivation.

1. **Removal of Linearly Dependent Search Functions:** In (25), we search $\mathcal{Y}^{(2)}$ for a minimal set of functions that span the function $\boldsymbol{\tau}$. However, for revolute joints, functions $\sin^2 q_i$, $\cos^2 q_i$, and 1 in $\mathcal{Z}^{(2)}_{q_i}$ are linearly correlated as $\sin^2 q_i + \cos^2 q_i - 1 = 0$. Thus, any function in $\mathcal{Y}^{(2)}$ containing $\sin^2 q_i$ could be rewritten as a linear combination of two other functions in $\mathcal{Y}^{(2)}$. Accordingly, any function in the search space which includes a squared sine term can safely be eliminated from the formulation, reducing the maximum matrix inversion size from $6^{n_r}\, 3^{n_p}$, as in Section 3.1.1, to $5^{n_r}\, 3^{n_p}$, since the effective size of $\mathcal{Z}^{(2)}_{\lambda}$, as in (6), has been reduced from 6 to 5.
2. **Reduced Gravity Term Search Space:** By Lemma 3, the gravity vector $\mathbf{G}(\mathbf{q})$ is guaranteed to be spanned by $\mathcal{Y}^{(1)} \subset \mathcal{Y}^{(2)}$. Therefore, all search functions in the gravity sub-regression problem which are not in this subset can be safely eliminated, which, as per (9), further reduces the gravity sub-regression problem to a search space of size $\left|\Upsilon^{(1)}\right|$, or $3^{n_r}\, 2^{n_p}$ functions.
3. **Reduced Velocity Term Search Space:** After solving for the base regressor terms of the *projected angular acceleration* functions $\mathcal{Y}^{(2)}_{\ddot{\mathbf{q}}}$, we form the set of functions $\mathcal{Y}_{p,\ddot{\mathbf{q}}}$, which spans the induced torques in the joints solely due to the angular acceleration of the joint coordinates, $\boldsymbol{\tau}_{\ddot{\mathbf{q}}}(\mathbf{q}, \ddot{\mathbf{q}}) = \mathbf{D}(\mathbf{q})\ddot{\mathbf{q}}$. Let the symbol $\Upsilon_{\ddot{\mathbf{q}}}$ denote the set of r distinct geometric ratio functions from $\Upsilon^{(2)}$ that are used to create the set $\mathcal{Y}_{p,\ddot{\mathbf{q}}}$. By Lemma 4, these functions will entrywise span the matrix $\mathbf{D}(\mathbf{q})$. Additionally, by Lemma 5, we know that the geometric multiplier functions of the velocity terms are formed by linear combinations of the derivatives of $\mathbf{D}(\mathbf{q})$. In other words, the space of search functions for the velocity terms, $\mathcal{Y}^{(2)}_{\dot{\mathbf{q}}}$, is limited to the (nr) terms associated with the n partial derivatives of the functions in $\mathcal{Y}_{p,\ddot{\mathbf{q}}}$, w.r.t each joint variable. Furthermore, the formula to determine these (rn) terms has already been derived, in (10) and (11). Because $p \ll t$, it is generally a significantly smaller problem to search the set of (nr) partial derivative functions, rather than the full set of $5^{n_r}\, 3^{n_p}$ otherwise required.
4. **Joint 1 Dependency Removal:** The position of the first joint is just a rotation or translation relative to gravity, but otherwise has no effects on the acceleration or velocity components of the dynamic model. Therefore, any projected angular or velocity acceleration function (e.g., $y_j \in \mathcal{Y}^{(2)}_{\ddot{\mathbf{q}}}$ or $y_j \in \mathcal{Y}^{(2)}_{\dot{\mathbf{q}}}$), which also includes any function of the variable q_1, can safely be removed from the search functions. This filter further reduces the maximum size of the linear system to be solved from $5^{n_r}\, 3^{n_p}$ to $5^{(n_r-1)}\, 3^{n_p}$, if joint 1 is revolute, and $5^{n_r}\, 3^{(n_p-1)}$, if joint 1 is prismatic.

After all the simplifications, the size of linear systems to be solved has been significantly reduced, from a single problem of size t, as in (21), to $(n+1)(n+2)/2$ more tractable sub-problems with maximum size

$$t = 6^{n_r} 3^{n_p}(n+1)(n+2)/2 \qquad \longrightarrow \qquad t' = \begin{cases} 5^{(n_r-1)}\, 3^{n_p} & \text{if joint 1 is revolute,} \\ 5^{n_r}\, 3^{(n_p-1)} & \text{if joint 1 is prismatic.} \end{cases} \tag{48}$$

3.2. Numerical Function Representation

The PSDM algorithm involves generating, manipulating, and evaluating a large number of projected acceleration functions in $\mathcal{Y}^{(2)}$. In this section, we discuss how these functions can be encoded and evaluated numerically in matrix form. This encoding lets us easily represent PSDM models, but also makes manipulating and altering the models possible with simple matrix operations. Because the form of the functions we wish to track is very regular, encoding them numerically is not overly complex. If we define the vector *generator function* $\boldsymbol{\gamma} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{5n}$ as

$$\boldsymbol{\gamma}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \begin{bmatrix} \mathbf{q}^\mathsf{T} & \sin(\mathbf{q})^\mathsf{T} & \cos(\mathbf{q})^\mathsf{T} & \dot{\mathbf{q}}^\mathsf{T} & \ddot{\mathbf{q}}^\mathsf{T} \end{bmatrix}^\mathsf{T}, \tag{49}$$

then a $5n$ integer *exponent vector* $\mathbf{e}_j$ can fully represent any function $y_j(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \in \mathcal{Y}^{(2)}$ as

$$y_j(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \prod_{h=1}^{5n} \Big[\gamma_h(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\Big]^{e_{h,j}}, \qquad e_{h,j} \in \{0, 1, 2\}, \tag{50}$$

where $e_{h,j}$ represents the integer in the h^{th} position of $\mathbf{e}_j$. To represent any set of functions such as $\mathcal{Y}^{(2)}$ or $\mathcal{Y}_p$, the vectors $\mathbf{e}_j$ can be assembled into an integer *exponent matrix* $\mathbf{E} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 & \cdots \end{bmatrix}$.

For the remainder of this paper, the following notation will be used. The calligraphic $\mathcal{Y}$ notation will be used to denote sets of functions (such as $\mathcal{Y}^{(2)}$ or $\mathcal{Y}_p$), and the bold $\mathbf{E}$ will be used to denote numeric exponent matrix representation of these sets, with the same annotations (e.g., $\mathbf{E}^{(2)}$ or $\mathbf{E}_p$).

Example 4. The $p = 10$ functions $y_j \in \mathcal{Y}_p$ defined in Example 3 for the two-link manipulator are

$$\begin{array}{lllll} y_1 = \ddot{q}_1, & y_2 = c_2\ddot{q}_1, & y_3 = \ddot{q}_2, & y_4 = c_2\ddot{q}_2, & y_5 = s_2\dot{q}_1^2, \\ y_6 = s_2\dot{q}_2^2, & y_7 = s_2\dot{q}_1\dot{q}_2, & y_8 = s_1 s_2 g, & y_9 = c_1 g, & y_{10} = c_1 c_2 g. \end{array}$$

This set of functions can be encoded using (50) with a generator vector $\boldsymbol{\gamma}$ and exponent matrix $\mathbf{E}_p$ defined as

$$\boldsymbol{\gamma} = \begin{bmatrix} q_1 \\ q_2 \\ s_1 \\ s_2 \\ c_1 \\ c_2 \\ \dot{q}_1 \\ \dot{q}_2 \\ \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix}, \qquad \mathbf{E}_p = \begin{array}{c} \begin{array}{cccccccccc} y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}. \qquad \triangleleft$$

3.3. Selection of Sample Points & Implicit Model Simplifications

The numerical analyses of both finding the base regressor functions $\mathcal{Y}_p$ in Section 2.2.1, as well as identifying the reduction matrices $\mathbf{P}_i$ in Section 2.2.2, require sampling of the inverse dynamics function to form the matrices $\mathbf{Y}_t$ and $\mathbf{T}_t$ as in (24), and $\mathbf{T}_i$ as in (31). Since this sampling is completely numerical, we can sample the function at any point of our choosing. To generate maximally exciting samples, linear dependencies in the samples should be avoided, and the scale of all samples should be similar. In the current work, this is achieved through the use of a uniform random distribution. For the joint samples, we sample $\mathbf{q}$ randomly from within the joint limits of the robot, and $\dot{\mathbf{q}}, \ddot{\mathbf{q}}$ in the range $[-1,\ 1]$. In the sampling of inertial parameters $\mathbf{X}$ to calculate the reduction matrices $\mathbf{P}_i$, we similarly sample from a uniform distribution in the range $[-1,\ 1]$.

An interesting feature of PSDM is that the results can be *implicitly simplified* by selecting the sample points of the linear systems in specific ways. Often, while the precise values of inertial parameters in $\mathbf{X}$ are not known for a robot, it may be known that certain parameters have a negligible effect on the dynamics. Alternatively, it could be desirable to find a model neglecting certain terms. For example, in the two-link planar manipulator example in Figure 1, we may wish to implicitly assume that all the inertial parameters except m_i, $r_{x,i}$, and $I_{zz,i}$ are negligible or zero. To achieve these simplifications, one simply needs to select sample points which satisfy the assumption one wishes to make. To assume an inertial parameter is negligible, simply set that parameter to zero in all samples. This will cause any effects of that parameter to be "invisible" to the algorithm, and the resulting simplifications to the model happen automatically. Indeed, *any* prescribed, linear relationship in the joint samples will be "detected" by the algorithm when finding the reduction matrices $\mathbf{P}_i$ and $\mathbf{B}_i$, the resulting equations will be simplified accordingly, and the size of the base parameters $\boldsymbol{\theta}_b$ may be reduced.

These implicit simplifications can lead to much simpler formulations for the dynamic models of manipulators, but are achieved entirely by numerical methods. With other formulations, these simplifications may not be as intuitive to make, or would require proper symbolic software to perform. Moreover, the simplifications are completely immune to the complexity of the models. Whereas symbolic simplifications can fail in some complicated situations, PSDM takes a "black box" approach – looking instead only at the linear relationship between input and outputs, and simplifying accordingly.

3.4. Solution to Linear Equations & Tolerancing

In the PSDM algorithm, linear systems must be solved at two points in the derivation, in (25) and (32). Many algorithms exist to solve systems of linear equations. The following section will briefly discuss the implementational considerations of these steps. Three issues arise here – we must solve a linear system with high accuracy, we must select an appropriate number of samples N to solve the linear system , and we must select an appropriate tolerance ε in (26) to identify the base regressor terms.

3.4.1. Solution to Linear Equations with Iterative Refinement

With PSDM, there is no noise in the "measurements" of the system identification problem. However, the effects of machine precision will perturb the resulting parameter matrix $\boldsymbol{\Theta}_t$ in (25), as well as $\boldsymbol{\Psi}_i$ in (32). If the perturbation to $\boldsymbol{\Theta}_t$ is too large, it can be difficult to use an appropriate tolerance ε in (26) to eliminate the uncorrelated functions in $\mathcal{Y}^{(2)}$, without accidentally removing functions which have a small, but non-negligible effect on the joint torques. Additionally, if the perturbation to $\boldsymbol{\Psi}_i$ is large, it will result in errors to the resulting reduction matrices $\mathbf{P}_i$ and $\mathbf{B}_i$, which will cause the final PDSM model to give skewed results. In the current work, we use double-precision 52-bit floating-point arithmetic in all computations. However, we alleviate the noise issue caused by machine precision by using the method of extra-precise iterative refinement, as described in detail by Demmel et al. [31] and Moler [32]. In this method, a basic solution is calculated to the system of linear equations using an LU or QR decomposition. Then, a high-precision residual is calculated using triple or quadruple precision math, and used to "refine" the original solution. This method involves only slightly more computations than the basic solution. The result, however, generally reduces the magnitude of the residual by $2 - 4$ orders of magnitude, and allows for solutions to the system, even if $\mathbf{Y}$ is somewhat ill-conditioned. Using this method, rather than the standard double-precision algorithms, makes it easier to eliminate functions in (26), and also results in more accurate reduction matrices $\mathbf{P}_i$ and $\mathbf{B}_i$, which in turn gives a more accurate PSDM model.

3.4.2. Number of Sample Points

The linear system of equations in Sections 2.2.1 and 2.2.2 can either be solved as an exact linear system, where $\mathbf{Y}$ is square and the number of samples N is equal to the number of functions t' or p, or as the least-squares solution to an over-determined system, where the number of samples N is greater than the number of functions and $\mathbf{Y}$ has more rows than columns. This choice is a trade-off of speed versus accuracy. Identifying the least-squares solution to a set of N linear equations in m variables using QR decomposition with householder reflections is of complexity $O(m^2N)$ where $N > m$. Alternatively, finding the exact solution to a square linear system using LU decomposition can be simplified slightly to a complexity of $O(N^{2.8})$ [33]. As such, the least-squares solution takes more time, particularly with larger problems. However, having more equations than unknowns helps decrease the condition number of the linear system, and can help reduce the effects of machine precision noise in the results of PSDM. In the current work, we use N equal to the number of functions t for the primary linear systems in Section 2.2.1, since in this step the systems of equations are large, and the accuracy of $\boldsymbol{\Theta}_t$ is less critical – we only need to identify which parameters are negligible. However, for the secondary numerical regression in Section 2.2.2, we use a least-squares solution with N equal to twice the number of functions p, in order to increase the accuracy of the resulting $\mathbf{P}_i$. For this step, the size of the regression problem is much smaller than in the initial regression problem in Section 2.2.1, so the added complexity of using a QR decomposition of a non-square system has a negligible effect on the overall speed of the PSDM derivation. However, better accuracy in the resulting matrices $\mathbf{P}_i$ directly effects the accuracy of the final PSDM model, after derivation.

3.4.3. Tolerancing

A final issue in the implementation of PSDM is the selection of the tolerance ε, which is used in (26) to determine if a function y_j has a non-negligible effect on the joint torques $\boldsymbol{\tau}$. In the current work, this tolerance was set heuristically to $\varepsilon = 10^{-10}$. This was found to give good results for all the models tested, however future work could investigate using more intelligent tolerancing based on the residuals of the linear system solutions and estimations of the condition number of $\mathbf{Y}_t$.

3.5. Inclusion of Additional Dynamic Effects

In the dynamic modeling of robots, it is often desirable to add “additional” effects to the canonical equations of motion, as in (1). Examples include the inertial effects of the motors and drives, friction in the joints, and more, depending on the application. It is possible for PSDM to handle these effects, with some modifications. Consider the inverse dynamics function of (1), with additional terms added to account for various additional physical phenomena,

$$\boldsymbol{\tau} = \mathbf{D}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \boldsymbol{\tau}_m + \boldsymbol{\tau}_{\text{fric}} + \dots \tag{51}$$

where $\boldsymbol{\tau}_m$ represents the effect of motor inertias, $\boldsymbol{\tau}_{\text{fric}}$ the effect of joint friction, and to which any number of additional effects, as required, could be applied. To be experimentally identified, the effects of $\boldsymbol{\tau}_m$, $\boldsymbol{\tau}_{\text{fric}}$, or any other factor must either be included in the PSDM model, or be possible, through various methods, to experimentally “isolate”, such that the additional effects can be quantified independently to the PSDM model.

The effect of the motor inertias is difficult to isolate experimentally from the rest of the inertial model, since the drives are rigidly connected to the inertias of the links themselves. Fortunately, however, this effect is easily integrated into the PSDM model itself. Let $I_{m,i}$ represent the equivalent inertia of the gearbox and motor armature of the i^{th} joint, after transmission ratios have been taken into account. The torque due to motor inertias is then determined as

$$\tau_{m,i} = I_{m,i}\,\ddot{q}_i, \qquad i = 1,\dots,n. \tag{52}$$

Since the function $\ddot{q}_i$ is already a member of the set $\mathcal{Y}^{(2)}$, PSDM can readily handle the addition of this effect. The only alteration required is that the sampling function (RNE method, or other), which is used to reduce $\mathcal{Y}^{(2)}$ to a minimal set, also includes these effects. If this is done, the PSDM algorithm will otherwise execute normally, except that the motor inertia effects will be integrated into the PSDM model, and the motor inertias themselves will be reduced into the resulting base inertial parameter set.

Other effects not directly related to the motion of the links are not so easily integrated into the PSDM model. Friction, for example, is highly nonlinear and the models used vary by application [34]. Fortunately, however, there exist ways to isolate these effects in an identification process, such that frictional models for each joint can be developed independently of the PSDM model. Some techniques which have been discussed in literature are sequential identification, where small parts of the dynamic model are identified one at a time using specialized excitation trajectories [22], or through spectrum analysis, where one leverages knowledge of the frequencies and phases of the additional dynamic effects to isolate them [35]. Other common joint effects, such as backlash, drive flexibility, etc., can also be separated experimentally through similar methods.

3.6. Summary of the PSDM Algorithm

With all the terms, formulae and considerations now covered, we can provide a summary of the key steps of the PSDM derivation in a more compact form. To begin, we assume that we have a serial manipulator, whose dynamics we wish to calculate. Moreover, we have a means of numerically sampling the inverse dynamics function of this manipulator, $\boldsymbol{\tau}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}},\mathbf{X})$, for any joint state $(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ and inertial parameters $\mathbf{X}$. This sampling method does not need to be efficient, since it is only used in the derivation step. It also does not need to be in regressor form. Here, we use the RNE algorithm, however any other dynamic model would suffice, for example dynamic modeling software, etc. Then, the derivation involves the following four steps:

1. **Identify the projected acceleration functions $\mathcal{Y}^{(2)}$ and $\mathbf{E}^{(2)}$:** From the joint types of the robot, generate the set of all functions in $\mathcal{Y}^{(2)}$ that are identified by Proposition 3 to be a basis for the inverse dynamics of the robot, as in (14). These functions will be numerically represented by a $5n \times t$ exponent matrix $\mathbf{E}^{(2)}$ and a generator vector $\boldsymbol{\gamma}$, as in Section 3.2.
2. **Pre-filter the search functions:** Remove all columns from $\mathbf{E}^{(2)}$ that cannot contribute to the inverse dynamic model, as per the conditions listed in Section 3.1.
3. **Identify the base regressor functions $\mathcal{Y}_p$ and $\mathbf{E}_p$:** First, split the matrix $\mathbf{E}^{(2)}$ into $(n+1)(n+2)/2$ sub-matrices, for each acceleration function $a \in \mathcal{A}$. For each sub-matrix, generate $\mathbf{Y}_{t,a}(\mathbf{q},\dot{\mathbf{q}},\ddot{\mathbf{q}})$ and

$\mathbf{T}_{t,a}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, \mathbf{X})$ as per (24) and (47), for some set of joint states and a single set of inertial parameters $\mathbf{X}$, sampled as in Section 3.3. Then, find $\boldsymbol{\Theta}_t$ as per (25), and eliminate all columns of $\mathbf{E}^{(2)}$ corresponding to zero rows of $\boldsymbol{\Theta}_t$, as per (26). The resulting reduced matrix is renamed $\mathbf{E}_p$ and numerically represents the set of base regressor functions, $\mathcal{Y}_p$.

4. **Find the reduction matrices $\mathbf{B}_i, \mathbf{P}_i$:** Generate a set of joint state samples and inertial parameter samples, as in Section 3.3. Form the matrices $\mathbf{Y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ as in (24) and $\mathbf{T}_i$ as in (31). Then, find the n inertial parameters matrices $\boldsymbol{\Psi}_i$ for each joint i as per (32). Stack these matrices vertically into $\boldsymbol{\Psi}_*$, then find $\mathbf{B}_i$, $\mathbf{P}_i$ and ℓ as per (38), (39), (41) and (42).

The inverse dynamic model of the robot can then be calculated using $\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ and $\mathbf{P}_i$ and (45) or (46), calibrated with an ℓ-vector of base inertial parameters $\boldsymbol{\theta}_b$.

Example 5 (PSDM derivation for a two-link planar manipulator)**.** The entire PSDM derivation can now be demonstrated in full using the two-link planar manipulator from Figure 1. Because the derivation requires numerical kinematic parameters, let us arbitrarily use link lengths of $\tilde{a}_1 = 1.2$, and $\tilde{a}_2 = 1.1$. The sets $\Upsilon^{(1)}$, $\Upsilon^{(2)}$, $\mathcal{Y}^{(1)}$, and $\mathcal{Y}^{(2)}$ have been defined previously in Examples 1 and 2. To find the set of *base regressor terms* $\mathcal{Y}_p$, we solve the $(n+1)(n+2)/2 = 6$ sub-regression problems associated with each acceleration $a \in \mathcal{A}$, as described in Section 3.1.1.

- *Gravity acceleration functions*: Only the $3^{n_r} = 9$ functions in $\mathcal{Y}^{(1)}$ need to be considered here. We generate 9 random poses in the range $q_1, q_2 \in [-\pi, \pi]$, and generate a single random inertial set of parameters with $m_i, I_{zz,i}, r_{x,i} \in [0.1, 1]$, and all other parameters identically zero, since we wish for them to be ignored in the analysis. Then, we form a 9×2 matrix $\mathbf{T}_{t,\mathbf{q}}$ as in (24) and (47); and a 9×9 matrix $\mathbf{Y}_{t,\mathbf{q}}$ as in (24). Solving for $\boldsymbol{\Theta}_{t,\mathbf{q}}$ gives a 9×2 matrix in which all but three rows are identically zero, corresponding to the functions
$$\mathcal{Y}_{p,\mathbf{q}} = \{ s_1 s_2 g, \quad c_1 g, \quad c_1 c_2 g \}.$$
- *Angular acceleration functions*: This process is repeated with the angular acceleration terms $\ddot{q}_1$ and $\ddot{q}_2$, and their associated $5^{n_r} = 25$ functions. Here, solving for $\boldsymbol{\Theta}_{t,\ddot{q}_1}$ and $\boldsymbol{\Theta}_{t,\ddot{q}_2}$ identifies only two non-zero rows for each, corresponding to the functions
$$\mathcal{Y}_{p,\ddot{q}_1} = \{ \ddot{q}_1, \quad c_2 \ddot{q}_1 \}, \qquad\qquad \mathcal{Y}_{p,\ddot{q}_2} = \{ \ddot{q}_2, \quad c_2 \ddot{q}_2 \}.$$
- *Velocity acceleration functions*: As described in Section 3.1.2, we know that the geometric ratio functions of the velocity terms are within the set of derivatives of the geometric ratio functions from the projected angular acceleration terms $\mathcal{Y}_{p,\ddot{\mathbf{q}}}$, with respect to q_1 and q_2. The geometric ratio functions found in $\mathcal{Y}_{p,\ddot{\mathbf{q}}}$ are 1 and c_2. Therefore, we only need to search the set of six functions obtained by the multiplication of the set of three possible velocity accelerations $\dot{q}_1^2$, $\dot{q}_2^2$ or $\dot{q}_1\dot{q}_2$ and the set of two possible partial derivative functions, $\{ \frac{\partial c_2}{\partial q_1}, \frac{\partial c_2}{\partial q_2} \} = \{ 1, \quad s_2 \}$. Combining these geometric multiplier functions with the three possible velocity accelerations $\dot{q}_1^2$, $\dot{q}_2^2$ and $\dot{q}_1\dot{q}_2$ yields the three sets of search functions, $\{ \dot{q}_1^2, \quad s_2\,\dot{q}_1^2 \}$, $\{ \dot{q}_2^2, \quad s_2\,\dot{q}_2^2 \}$, and $\{ \dot{q}_1\dot{q}_2 \quad s_2\,\dot{q}_1\dot{q}_2 \}$, respectively. Reducing these equations as before reveals only one necessary function in each set, corresponding to
$$\mathcal{Y}_{p,\dot{q}_1^2} = \{ s_2\,\dot{q}_1^2 \}, \qquad \mathcal{Y}_{p,\dot{q}_2^2} = \{ s_2\,\dot{q}_2^2 \}, \qquad \mathcal{Y}_{p,\dot{q}_1\dot{q}_2} = \{ s_2\,\dot{q}_1\dot{q}_2 \}.$$

Concatenating the functions associated with each of the gravity, angular acceleration and velocity accelerations gives the $p = 10$ *base regressor functions* in $\mathcal{Y}_p$ and the corresponding regression matrix
$$\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \begin{bmatrix} \ddot{q}_1 & c_2\ddot{q}_1 & \ddot{q}_2 & c_2\ddot{q}_2 & s_2\dot{q}_1^2 & s_2\dot{q}_2^2 & s_2\dot{q}_1\dot{q}_2 & s_1 s_2 g & c_1 g & c_1 c_2 g \end{bmatrix}, \tag{53}$$
which can be verified by inspection to be the same as the functions identified previously in Example 3. The next step in the derivation is the secondary numerical analysis to find the reduction matrices $\mathbf{P}_1$ and $\mathbf{P}_2$, which reduce the problem to a set of base inertial parameters. We generate at least $p = 10$ random joint states $(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j)$, as before, and at least $10n = 20$ random inertial parameter sets $\mathbf{X}_k$, all sampled randomly in the same ranges as the primary analysis. The matrices $\mathbf{Y}_p$ and $\mathbf{T}_i$ are computed as in (31), then the matrices $\mathbf{P}_1$

and $\mathbf{P}_2$ are calculated using (34), (37) and (38) as

$$\mathbf{P}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -2.4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1.2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.2 \\ 0 & 0 & 0 & 2.4 \end{bmatrix}, \qquad \mathbf{P}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1.2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{54}$$

which map the $p = 10$ base regressor functions to a set of $\ell = 4$ *base inertial parameters*

$$\boldsymbol{\theta}_b = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 \end{bmatrix}^{\mathsf{T}}.$$

The matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ clearly show the linear dependencies between $\boldsymbol{\theta}_b$ and $\mathbf{y}_p$, and are sparse, composed primarily of zeros and ones. The remaining numbers are simple ratios of kinematic parameters. In this case, by inspection, they can be identified as $1.2 = \tilde{a}_1$ and $2.4 = 2\,\tilde{a}_1$. The base inertial parameters $\boldsymbol{\theta}_b$ will be some unknown function of the kinematic and inertial parameters. However, explicit knowledge of this function is not necessary since $\boldsymbol{\theta}_b$ should be identified experimentally from test data. Alternatively, $\boldsymbol{\theta}_b$ can also be obtained with a numerically sampled $\boldsymbol{\tau}$, given knowledge of a known set of full inertial parameters, using (45). The entire PSDM algorithm to derive this result did involve several steps, however every step is completely procedural and therefore can be completed quickly via computer. Once programmed and run, from the input of a DH table, this process completes for the current example in approximately 11 ms on a personal computer. Interested readers can find links in the Supplementary Material section to a full Matlab implementation of the PSDM algorithm. Sample code implementing this example is provided there and allows the reader to follow along with real numerical examples. $\triangleleft$

4. Extensions

To this point, we have discussed the proofs, methodology, and implementation of PSDM for use in deriving the inverse dynamic model of a robotic manipulator. In this section, we demonstrate three extensions to PSDM which allow for 1) easily generated optimized real-time code; 2) generation of a forward dynamic model; and 3) leveraging the organized structure of PSDM to derive slightly less accurate, but more computationally efficient dynamic models for use in constrained computational environments, such as high rate control loops or embedded hardware.

4.1. Model Optimization & Real-Time Code Generation

For real-time purposes, the inverse dynamic PSDM model

$$\boldsymbol{\tau} = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \begin{bmatrix} \mathbf{P}_1\boldsymbol{\theta}_b & \cdots & \mathbf{P}_n\boldsymbol{\theta}_b \end{bmatrix} = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \boldsymbol{\Phi}_b \tag{55}$$

must be calculated with a low computational load. Section 3.2 identifies a convenient numerical means of representing the many functions that must be kept track of and evaluated during the PSDM derivation, through a matrix $\mathbf{E}$ and a generator vector $\boldsymbol{\gamma}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. However, this representation should not be used for real-time purposes, as it involves significant unnecessary indexing, exponentiation by zero and 1, and multiplication by unity. Compared to the other computationally expensive tasks in a PSDM derivation, this overhead is more or less negligible. However, for real-time evaluation, it should be avoided.

Fortunately, the very regular form of a PSDM model, represented by $\mathbf{E}$ and $\mathbf{P}_i$, makes optimization of the inverse dynamic model straightforward. We leverage the regular form of the PSDM model to write out C code procedurally for the model. With $\boldsymbol{\gamma}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ pre-computed for each exponent $\{1, 2\}$ into variables `g1` and `g2`, we can transcode any function y_j, represented by a column of $\mathbf{E}$, into a string of C-code. For example, a function y_j, represented by a column vector $\mathbf{e}_j$ as below, could be transcoded as

$$\mathbf{e}_j = \begin{bmatrix} 0 & 1 & 0 & 2 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}^{\mathsf{T}} \quad \Longrightarrow \quad \texttt{y[j] = g1[1]*g2[3]*g1[6]*g1[7];}$$

This step removes the overhead from unnecessary exponentiation and multiplication by unity. However, a significant number of redundant multiplications remain in the calculation of the regressor $\mathbf{y}_p$, since many of the individual functions contain common terms. For example, consider the two functions in Example 5, $y_9 = c_1 g$ and $y_{10} = c_1 c_2 g$. Both functions require multiplying c_1 and g. Implemented as above, this would require three multiplications, but with y_{10} rewritten as $y_{10} = c_2 y_9$, it is possible to achieve the same result with only two multiplications. For more complex models, the reduction in operations become much greater.

To minimize the number of operations in the dynamic model, we use the following strategy. If one expands the inverse dynamic model (55) using the definitions of $\mathbf{E}$ and $\boldsymbol{\gamma}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ in (49) and (50), the calculation for each joint i becomes the sum of p terms

$$\tau_i = \Phi_{1,i}\, {\gamma_1}^{e_{1,1}}\, {\gamma_2}^{e_{2,1}} \cdots {\gamma_{5n}}^{e_{5n,1}} \;+\; \Phi_{2,i}\, {\gamma_1}^{e_{1,2}}\, {\gamma_2}^{e_{2,2}} \cdots {\gamma_{5n}}^{e_{5n,2}} \;+\; \cdots \;+\; \Phi_{p,i}\, {\gamma_1}^{e_{1,p}}\, {\gamma_2}^{e_{2,p}} \cdots {\gamma_{5n}}^{e_{5n,p}}, \quad (56)$$

where $\Phi_{j,i}$ is the number in the j^{th} row and i^{th} column of $\boldsymbol{\Phi}_b$. Consider "splitting" each j^{th} term of the summation above after the first element of $\boldsymbol{\gamma}$, as

$$\Phi_{j,i}\; {\gamma_1}^{e_{1,j}}\; {\gamma_2}^{e_{2,j}}\; \cdots\; {\gamma_{5n}}^{e_{5n,j}} = \left(\Phi_{j,i}\; {\gamma_1}^{e_{1,j}}\right) \cdot \left({\gamma_2}^{e_{2,j}}\; \cdots\; {\gamma_{5n}}^{e_{5n,j}}\right) = \left(\Phi_{j,i}\; {\gamma_1}^{e_{1,j}}\right) y_j^{\{1\}}.$$

Because the second part of each term, $y_j^{\{1\}} = \left({\gamma_2}^{e_{2,j}}\; \cdots\; {\gamma_{5n}}^{e_{5n,j}}\right)$, has been simplified compared to the original function y_j, the number of unique values of $y_j^{\{1\}}$ over all the functions in $\mathcal{Y}_p$ is reduced. However, we can calculate τ_i from this reduced set by grouping the terms $\left(\Phi_{j,i}\; {\gamma_1}^{e_{1,j}}\right)$ for each unique sub-function $y_j^{\{1\}}$. Let $\Phi_{j,i}^{\{1\}}$ be defined as the sum of the terms $\left(\Phi_{j,i}\; {\gamma_1}^{e_{1,j}}\right)$ over the duplicates of $y_j^{\{1\}}$,

$$\Phi_{j,i}^{\{1\}} = \sum_{h \in H} \Phi_{h,i} {\gamma_1}^{e_{1,h}}, \qquad\qquad H = \left\{ h \in \mathbb{N} \;\middle|\; y_h^{\{1\}} = y_j^{\{1\}} \right\}.$$

The torque τ_i can be calculated as a new linear combination of the reduced set of p' unique functions $y_j^{\{1\}}$,

$$\tau_i = \Phi_{1,i}^{\{1\}}\, y_1^{\{1\}} \;+\; \Phi_{2,i}^{\{1\}}\, y_2^{\{1\}} \;+\; \cdots \;+\; \Phi_{p',i}^{\{1\}}\, y_{p'}^{\{1\}}. \quad (57)$$

Each of the original elements ${\gamma_1}^{e_{1,j}}$ can take on one of 3 values, for each possible exponent $e_{1,j} \in \{0,\, 1,\, 2\}$. Accordingly, calculating (57) involves up to three times fewer terms than the original formula (56). Moreover, (57) is in the exact same form as (56), after one reduction step. Therefore, we can apply this process recursively to further reduce the number of multiplications. This recursion can be done a total of $5n$ times, for each element γ_h for $h = 1, \ldots, 5n$. After the $5n^{\text{th}}$ recursion, only a single unique function $y_1^{\{5n\}}$ will remain, and the joint torque is simply

$$\tau_i = \Phi_{1,i}^{\{5n\}}\, y_1^{\{5n\}}. \quad (58)$$

In this process, we can additionally exploit the sparsity of $\boldsymbol{\Phi}_b$ (or $\mathbf{P}_i$) by eliminating any multiplications by 0 or ± 1. This process can significantly reduce the computational load of the algorithm and is convenient to implement because of the regular form of the PSDM model. The reduction in multiplications and additions can be quite significant, particularly in more complex dynamic models where the computation load can be reduced by over 90%. This reduction in operations is quantified for some example manipulators in Section 5. Additionally, further examples of this procedure are given in the supplementary material provided, including Matlab code that fully implements this process for any PSDM model.

4.2. Forward Dynamic Modeling with PSDM

A forward dynamic model, where instead one assumes known torques and solves for the resulting joint accelerations, is useful in many applications, in particular in simulation, but is also used in many control algorithms. As with the inverse dynamic model, it is important to have an accurate dynamic model, which is also computationally efficient. Fortunately, the organized structure of a PSDM model offers a simple means of deriving the forward dynamic model simply through rearrangements of the inverse dynamic model. Additionally, the forward model can be optimized in the same way as Section 4.1 and will use the same set of base inertial parameters as the inverse model.

After the PSDM derivation has been run for a given manipulator, we will have a set of base regressor functions $\mathcal{Y}_p$, represented in a $5n \times p$ integer matrix $\mathbf{E}_p$, as well as n reduction matrices $\mathbf{P}_i$ of dimension $p \times \ell$. To compute the forward dynamics, we need to calculate two separate quantities: the mass matrix $\mathbf{D}(\mathbf{q})$, and the induced torques $\boldsymbol{\tau}_{\text{ind}}(\mathbf{q}, \dot{\mathbf{q}})$ due to Coriolis, centrifugal, and gravity effects. Then, we can determine the resulting joint accelerations through rearrangements of (1) as

$$\ddot{\mathbf{q}} = \mathbf{D}(\mathbf{q})^{-1} \Big[\boldsymbol{\tau} - \boldsymbol{\tau}_{\text{ind}}(\mathbf{q}, \dot{\mathbf{q}}) \Big] = \mathbf{D}(\mathbf{q})^{-1} \Big[\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \dot{\mathbf{q}} - \mathbf{G}(\mathbf{q}) \Big]. \quad (59)$$

First, we formally define the following variables.

$\mathcal{Y}_{\ddot{q}_i}$: The subsets of $\mathcal{Y}_p$, for each joint i, which involve scaling of the angular acceleration $\ddot{q}_i$,

$$\mathcal{Y}_{\ddot{q}_i} = \left\{ y \in \mathcal{Y}_p \;\middle|\; y(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \ddot{q}_i \, \upsilon(\mathbf{q}), \;\; \upsilon(\mathbf{q}) \in \Upsilon^{(k)} \right\}. \tag{60}$$

We also define $\mathbf{y}_{\ddot{q}_i}(\mathbf{q}, \ddot{\mathbf{q}})$ as the vector function formed by the horizontal concatenation of the functions in $\mathcal{Y}_{\ddot{q}_i}$, in the same way as $\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is defined in (50), but for the reduced set $\mathcal{Y}_{\ddot{q}_i}$.

$\mathbf{P}_{i,\ddot{q}_i}$: The set of n matrices formed by keeping only the rows of $\mathbf{P}_i$ that correspond to elements of $\mathcal{Y}_{\ddot{q}_i}$, such that the number of rows in $\mathbf{P}_{i,\ddot{q}_i}$ is the same as the number of columns in $\mathbf{y}_{\ddot{q}_i}(\mathbf{q}, \ddot{\mathbf{q}})$.

$\mathcal{Y}_{\cancel{\ddot{\mathbf{q}}}}$: The set of all functions of $\mathcal{Y}_p$ which involve scaling of a centrifugal, Coriolis, or gravity acceleration. Equivalently, the set of all functions in $\mathcal{Y}_p$ that are not in $\mathcal{Y}_{\ddot{q}_i}$ for $i = 1, \ldots, n$,

$$\mathcal{Y}_{\cancel{\ddot{\mathbf{q}}}} = \left\{ y_j \in \mathcal{Y}_p \;\middle|\; y_j \notin \mathcal{Y}_{\ddot{q}_i}, \;\; i = 1, \ldots, n \right\}. \tag{61}$$

We also define the vector function $\mathbf{y}_{\cancel{\ddot{\mathbf{q}}}}(\mathbf{q}, \dot{\mathbf{q}})$ in the same manner as $\mathbf{y}_{\ddot{q}_i}(\mathbf{q}, \ddot{\mathbf{q}})$.

$\mathbf{P}_{i,\cancel{\ddot{\mathbf{q}}}}$: The set of n matrices formed the same way as $\mathbf{P}_{i,\ddot{q}_i}$, but for the set $\mathcal{Y}_{\cancel{\ddot{\mathbf{q}}}}$ rather than $\mathcal{Y}_{\ddot{q}_i}$.

Defining these variables separates the induced torques of the velocity and gravity effects from the effects of each joint acceleration. Numerically, these sets and matrices are trivially computed, since, in the numerical representation $\mathbf{E}_p$ of $\mathcal{Y}_p$, columns associated with each subset above can be found by checking for non-zero exponents in the rows corresponding to each acceleration term $\ddot{q}_i$ in the generator vector $\boldsymbol{\gamma}$, as in (49).

From these definitions, the induced torque $\boldsymbol{\tau}_{\text{ind}}(\mathbf{q}, \dot{\mathbf{q}})$ is calculated as

$$\boldsymbol{\tau}_{\text{ind}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{y}_{\cancel{\ddot{\mathbf{q}}}}(\mathbf{q}, \dot{\mathbf{q}}) \begin{bmatrix} \mathbf{P}_{1,\cancel{\ddot{\mathbf{q}}}} \boldsymbol{\theta}_b & \cdots & \mathbf{P}_{n,\cancel{\ddot{\mathbf{q}}}} \boldsymbol{\theta}_b \end{bmatrix} = \mathbf{y}_{\cancel{\ddot{\mathbf{q}}}}(\mathbf{q}, \dot{\mathbf{q}}) \, \boldsymbol{\Phi}_{\cancel{\ddot{\mathbf{q}}}}, \qquad \boldsymbol{\Phi}_{\cancel{\ddot{\mathbf{q}}}} \triangleq \begin{bmatrix} \mathbf{P}_{1,\cancel{\ddot{\mathbf{q}}}} \boldsymbol{\theta}_b & \cdots & \mathbf{P}_{n,\cancel{\ddot{\mathbf{q}}}} \boldsymbol{\theta}_b \end{bmatrix}. \tag{62}$$

Additionally, the mass matrix $\mathbf{D}(\mathbf{q})$ is

$$\mathbf{D}(\mathbf{q}) = \begin{bmatrix} \mathbf{y}_{\ddot{q}_1}(\mathbf{q}, \mathbf{1}_1) \, \mathbf{P}_{1,\ddot{q}_1} \boldsymbol{\theta}_b & \cdots & \mathbf{y}_{\ddot{q}_n}(\mathbf{q}, \mathbf{1}_n) \, \mathbf{P}_{1,\ddot{q}_n} \boldsymbol{\theta}_b \\ \vdots & \ddots & \vdots \\ \mathbf{y}_{\ddot{q}_1}(\mathbf{q}, \mathbf{1}_1) \, \mathbf{P}_{n,\ddot{q}_1} \boldsymbol{\theta}_b & \cdots & \mathbf{y}_{\ddot{q}_n}(\mathbf{q}, \mathbf{1}_n) \, \mathbf{P}_{n,\ddot{q}_n} \boldsymbol{\theta}_b \end{bmatrix}, \tag{63}$$

where $\mathbf{1}_i$ is an n-vector with a 1 in the i^{th} position, and zeros elsewhere. Moreover, if we define the matrices $\boldsymbol{\Phi}_{\ddot{q}_i}$ similarly to $\boldsymbol{\Phi}_b$, this equation simplifies to

$$\mathbf{D}(\mathbf{q}) = \begin{bmatrix} \mathbf{y}_{\ddot{q}_1}(\mathbf{q}, \mathbf{1}_1) \, \boldsymbol{\Phi}_{\ddot{q}_1} \\ \vdots \\ \mathbf{y}_{\ddot{q}_n}(\mathbf{q}, \mathbf{1}_n) \, \boldsymbol{\Phi}_{\ddot{q}_n} \end{bmatrix}^{\mathsf{T}}, \qquad \boldsymbol{\Phi}_{\ddot{q}_i} \triangleq \begin{bmatrix} \mathbf{P}_{1,\ddot{q}_i} \boldsymbol{\theta}_b & \cdots & \mathbf{P}_{n,\ddot{q}_i} \boldsymbol{\theta}_b \end{bmatrix}. \tag{64}$$

The forward dynamic model can now be computed using (59). Similar to the inverse dynamic model, this entire process is completely numerical and procedural, and a computer can complete all these tasks quickly.

Additionally, generating real-time code as in Section 4.1 can be done by a nearly identical procedure to the inverse dynamic model, since the forward dynamic problem is composed of $n+1$ smaller inverse dynamic problems — one for the induced torques $\boldsymbol{\tau}_{\text{ind}}$, and n for each of the n column of the mass matrix $\mathbf{D}(\mathbf{q})$. Moreover, since $\mathbf{D}(\mathbf{q})$ is known to be positive definite, only the upper triangle of the matrix needs to be computed, and the solution to the matrix inversion in (59) can be sped up by using a Cholesky decomposition [36].

Example 6 (Forward dynamic model of a two-link planar manipulator)**.** The vector functions $\mathbf{y}_{\ddot{q}_i}$ and $\mathbf{y}_{\cancel{\ddot{\mathbf{q}}}}$ for the two-link planar manipulator are defined from the terms $\mathbf{y}_p$ listed previously, as

$$\mathbf{y}_{\ddot{q}_1} = \begin{bmatrix} \ddot{q}_1 & c_2 \ddot{q}_1 \end{bmatrix}, \qquad \mathbf{y}_{\ddot{q}_2} = \begin{bmatrix} \ddot{q}_2 & c_2 \ddot{q}_2 \end{bmatrix}, \qquad \mathbf{y}_{\cancel{\ddot{\mathbf{q}}}} = \begin{bmatrix} s_2 \dot{q}_1^2 & s_2 \dot{q}_1^2 & s_2 \dot{q}_1 \dot{q}_2 & s_1 s_2 g & c_1 g & c_1 c_2 g \end{bmatrix}.$$

The matrices $\mathbf{P}_{i,\ddot{q}_i}$ and $\mathbf{P}_{i,\cancel{\ddot{\mathbf{q}}}}$ are similarly defined for $i = 1, 2$, and, when multiplied with $\boldsymbol{\theta}_b$, give

$$\boldsymbol{\Phi}_{\ddot{q}_1} = \begin{bmatrix} \theta_2 & \theta_3 \\ -2\,\tilde{a}_1\,\theta_4 & -\tilde{a}_1\,\theta_4 \end{bmatrix}, \qquad \boldsymbol{\Phi}_{\ddot{q}_2} = \begin{bmatrix} \theta_3 & \theta_3 \\ -\tilde{a}_1\,\theta_4 & 0 \end{bmatrix}, \qquad \boldsymbol{\Phi}_{\cancel{\ddot{\mathbf{q}}}} = \begin{bmatrix} \theta_1 & \theta_4 & -\theta_4 & 0 & \tilde{a}_1\,\theta_4 & 2\,\tilde{a}_1\,\theta_4 \\ 0 & \theta_4 & -\theta_4 & -\tilde{a}_1\,\theta_4 & 0 & 0 \end{bmatrix}^{\mathsf{T}}.$$

The mass matrix $\mathbf{D}(\mathbf{q})$ is found from (64) as

$$\mathbf{D}(\mathbf{q}) = \begin{bmatrix} \theta_2 - 2\,\tilde{a}_1\,c_2\,\theta_4 & \theta_3 - \tilde{a}_1\,c_2\,\theta_4 \\ \theta_3 - \tilde{a}_1\,c_2\,\theta_4 & \theta_3 \end{bmatrix}. \qquad \triangleleft$$

4.3. Dynamic Model Complexity Reduction

In some applications, the computations may be limited to the point where it is not possible to evaluate the full dynamic model without additional simplification. In these cases, it is possible to neglect some of the terms of the dynamic model, and obtain a non-exact formulation which can be computed more quickly. This is not a new idea – Armstrong et al. [37], Izaguirre and Paul [38] and others have proposed similar methodologies for increasing the dynamic model performance. Others, such as Izaguirre et al. [39], have proposed asynchronous computation of model elements to allow for a similar increase in performance. In the current work, we simply wish to show that such simplifications can be accomplished in a straightforward and numerical manner with the PSDM algorithm, without resorting to symbolic methods.

In the PSDM algorithm, we sample the dynamic model and seek to identify the functions within $\mathcal{Y}^{(2)}$ that accurately fit the model. Because $\mathcal{Y}^{(2)}$ is guaranteed to form a basis for $\boldsymbol{\tau}$, this identification is exact. However, a natural progression of PSDM is simply to remove this requirement for exactness and keep only the functions in $\mathcal{Y}_p$ that are required to achieve the accuracy minimally required by the application. Consider a dynamic model of a given manipulator represented by a set of functions $\mathcal{Y}_p$, represented numerically in a matrix $\mathbf{E}_p$, a set of reduction matrices $\mathbf{P}_i$, and a vector of base inertial parameters $\boldsymbol{\theta}_b$ obtained either experimentally or numerically. Then, suppose we have a set of N joint states $(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j)$, which are representative of the task which the robot will be required to perform. This could, for example, be a set of joint angles $\mathbf{q}$ that occur in the active workspace of the robot, and the joint speeds $\dot{\mathbf{q}}$ and accelerations $\ddot{\mathbf{q}}$ could be sampled randomly around the upper limits of the allowable joint speeds.

For each joint sample j, we can compute an $n \times p$ matrix of the contributions of each function $y_k \in \mathcal{Y}_p$,

$$\boldsymbol{\tau}(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) = \Big[\boldsymbol{\tau}_{y_1}(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) \quad \cdots \quad \boldsymbol{\tau}_{y_p}(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) \Big] = \Big[\, \mathbf{y}_p(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j)_{\times n} \, \Big] \odot \boldsymbol{\Phi}_b^{\mathsf{T}}, \tag{65}$$

for $j = 1, \ldots, N$, where $\boldsymbol{\Phi}_b$ is as defined in (46), $\big[(\,\cdot\,)_{\times n}\big]$ represents the vertical concatenation of the same matrix n times, and the $\odot$ symbol represents element-wise multiplication. We then define the *relative normed contribution* $\varrho_{jk} \in [0, 1]$ of each for each function $y_k \in \mathcal{Y}_p$ for each sample j as

$$\varrho_{jk} = \frac{\big\| \boldsymbol{\tau}_{y_k}(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) \big\|}{\big\| \boldsymbol{\tau}(\mathbf{q}_j, \dot{\mathbf{q}}_j, \ddot{\mathbf{q}}_j) \big\|}. \tag{66}$$

A statistical analysis of the contribution of these functions over an appropriately sampled set of joint states can give one a measure of which functions contribute significantly to the dynamic model, and which ones have a near-negligible contribution. Moreover, if the samples $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ are particularly constrained – for example, if they are only in a particular region of the robot's workspace, or if the joint speeds and accelerations have well-known limits, then the contributions ϱ_{jk} will vary significantly over the functions y_k, that is to say, some functions will contribute very negligibly to the dynamic model.

To illustrate this, consider the 6-DOF spherical-wrist KUKA Agilus KR6 manipulator, with a 3 kg end-effector. The full inertial and kinematic parameters for this robot are given later in Table 1. The PSDM model of this robot has $p = 6086$ unique functions $y_k \in \mathcal{Y}_p$. Additionally, consider two "speed profiles":

1. ***Slow* speed profile:** $\mathbf{q} \in [-\pi, \pi]$ rad, $\dot{\mathbf{q}} \in [-0.1, 0.1]$ rad/s, and $\ddot{\mathbf{q}} \in [-1, 1]$ rad/s^2; and
2. ***Fast* speed profile:** $\mathbf{q} \in [-\pi, \pi]$ rad, $\dot{\mathbf{q}} \in [-1, 1]$ rad/s, and $\ddot{\mathbf{q}} \in [-10, 10]$ rad/s^2.

We compute the relative contributions ϱ_{jk} for each function $y_k \in \mathcal{Y}_p$ and for $N = 10^5$ samples. The frequency distribution of the contributions of these functions is plotted in Figure 2, considering both the median and maximum contribution of each function over the joint state samples. This figure shows that, even with a very general sampling of joint states, a significant number of the functions contribute negligibly to the overall dynamic model. Were we to further constrict the sampling to a smaller area in the workspace, even more functions could be considered negligible.

Given this, it is reasonable to consider simplifications of the model which involve neglecting functions with the lowest relative contributions. The corresponding columns from the matrices $\mathbf{P}_i$ can then also be eliminated. Moreover, if, during this removal of rows, any of the columns of $\mathbf{P}_i$ become identically zero for all joints i, then these columns can also be eliminated, and the corresponding elements of the base inertial parameters $\boldsymbol{\theta}_b$ can be

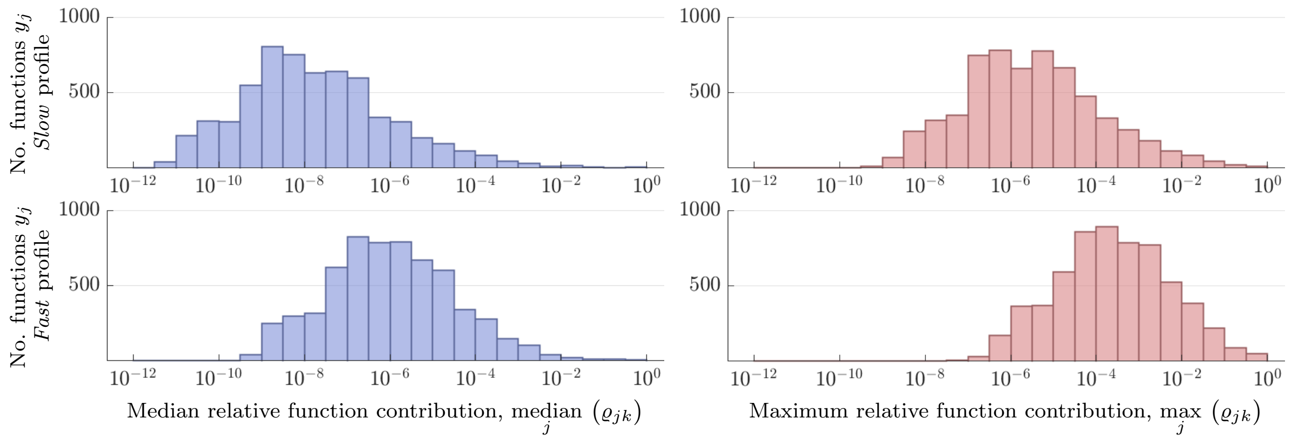


Figure 2: Frequency distribution of the median and maximum relative normed contribution ϱ_k of functions in $y_k \in \mathcal{Y}_p$ for a KUKA Agilus KR6 6-DOF manipulator, as evaluated from $N = 10^5$ uniformly distributed samples within the ranges for the *slow* speed profile (top) and the *fast* speed profile (bottom).

removed. The result is a reduced model defined by the tuple $(\hat{\mathbf{E}}, \hat{\mathbf{P}}_i, \hat{\boldsymbol{\theta}}_b)$ and an approximated inverse dynamics function $\hat{\boldsymbol{\tau}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$. This reduced model is more efficient to compute than the full model, and the reduction in accuracy may not affect the results of a controller depending on the accuracy required. Moreover, the reduced model is often more accurate than may be predicted by summing the contributions of the eliminated functions, since a regression for $\hat{\boldsymbol{\theta}}_b$ will give slightly different values than $\boldsymbol{\theta}_b$, tweaking the parameter vector to more accurately fit the model given omissions from the model.

This complexity reduction can equally be applied to the forward dynamic model, in the same manner. However, for the forward model, care must be taken, as inaccuracies in $\mathbf{D}(\mathbf{q})$ can disproportionally affect the accuracy of the model, due to the matrix inversion step. Instead, we recommend prioritizing acceleration functions in the model, and removing velocity or gravity functions before removing acceleration functions, as the error in acceleration estimates will have a much larger impact on the final model error.

The analysis presented here was not overly complex, and more advanced analyses of this trade-off between complexity and accuracy are possible. A more detailed analysis into the bandwidth of each function, as in the work of Kirćanski et al. [40], or investigating into evaluating each function y_k at a control rate proportional to its relative effect and its bandwidth similar to Izaguirre et al. [39], would likely produce even better results. In the current work, we simply wish to illustrate the ease by which one can perform these analyses numerically using PSDM, by leveraging the organized form of the derived models. With other symbolic routines, similar results could be achieved, however not with as much ease and likely requiring the use of symbolic routines. Further numerical examples will be provided in the following section, which presents several benchmarks for the PSDM algorithm.

5. Examples and Benchmarking

The PSDM algorithm is implemented in the Matlab environment and compiled into C. A recursive Newton-Euler (RNE) algorithm is similarly implemented as per Spong and Vidyasagar [27] and used as the sampling function for PSDM. We use this implementation to benchmark the PSDM algorithm's derivation time, evaluation time (for both inverse and forward models), and simplified computational models. In this benchmarking, we will make use of the following sample robotic manipulators.

1. *KUKA Agilus KR6 R700 manipulator:* A 6-DOF, spherical-wrist manipulator, with 3 kg non-symmetrical end-effector defined with appropriate, but arbitrary, non-zero inertial properties;
2. *KUKA LBR7 manipulator:* A 7-DOF manipulator, with the same end-effector as the Agilus robot;
3. *Fanuc SR-6iA SCARA manipulator:* To show the application of PSDM to a prismatic joint robot, we include a common implementation of the SCARA manipulator.

These manipulators are shown in Figure 3, and their kinematic and full inertial parameters $\mathbf{X}$ are given in Table 1. The inertial parameters are approximate and obtained from CAD models, except for the LBR7 robot, where experimental values from Stürz et al. [41] are used. For the purposes of demonstration, however, the accuracy of these parameters is not important – only that they are the correct order of magnitude.

A PSDM derivation is performed for each of the benchmark manipulators, as well as the two-link planar manipulator from earlier examples. The parameters and results of these derivations are summarized in Table 2.

Table 1: Kinematic & inertial parameters of benchmark manipulators.

(a) KUKA Agilus KR6 R700 6-DOF manipulator.

#	$\tilde{\theta}_i$	$\tilde{d}_i$	$\tilde{a}_i$	$\tilde{\alpha}_i$	m_i	$r_{x,i}$	$r_{y,i}$	$r_{z,i}$	$I_{xx,i}$	$I_{yy,i}$	$I_{zz,i}$	$I_{xy,i}$	$I_{xz,i}$	$I_{yz,i}$
1	q_1	183	25	$-\pi/2$	7.60	-37.6	43.5	-7.6	0.0430	0.0449	0.0490	0.0125	0.0022	-0.0024
2	q_2	0	-315	0	2.80	155.4	-6.2	114.8	0.0060	0.0393	0.0415	-0.0004	-0.0012	0.0001
3	q_3	0	-35	$\pi/2$	7.00	45.0	-7.0	4.0	0.0278	0.0375	0.0309	0.0010	0.0005	-0.0002
4	q_4	365	0	$-\pi/2$	2.90	-1.0	102.1	-2.8	0.0138	0.0058	0.0123	-0.0010	0.0002	0.0002
5	q_5	0	0	$\pi/2$	0.94	0.1	-30.0	10.6	0.0009	0.0010	0.0006	0.0001	0.0010	0.0001
6	q_6	80	0	0	3.00	-2.0	-19.0	67.0	0.0045	0.0050	0.0060	0.0001	-0.0003	0.0004

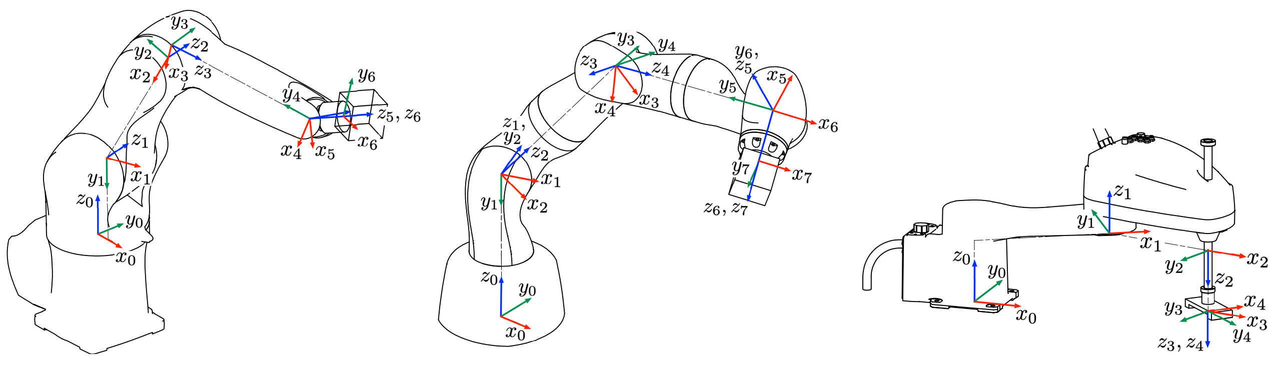
(b) Kuka LBR7 7-DOF manipulator. Inertial parameters taken from [41].

#	$\tilde{\theta}_i$	$\tilde{d}_i$	$\tilde{a}_i$	$\tilde{\alpha}_i$	m_i	$r_{x,i}$	$r_{y,i}$	$r_{z,i}$	$I_{xx,i}$	$I_{yy,i}$	$I_{zz,i}$	$I_{xy,i}$	$I_{xz,i}$	$I_{yz,i}$
1	q_1	340	0	$-\pi/2$	3.95	-3.5	1.6	-31.4	0.0046	0.0045	0.0003	0.0000	0.0000	0.0000
2	q_2	0	0	$\pi/2$	4.50	-7.7	166.8	-3.6	0.0003	0.0001	0.0004	0.0000	0.0000	0.0000
3	q_3	400	0	$\pi/2$	2.45	-2.2	-34.9	-26.5	0.0022	0.0022	0.0007	-0.0001	0.0001	0.0001
4	q_4	0	0	$-\pi/2$	2.61	0.2	-52.7	38.2	0.0384	0.0114	0.0499	0.0009	-0.0011	-0.0011
5	q_5	400	0	$-\pi/2$	3.41	0.1	-2.4	-211.3	0.0028	0.0028	0.0001	0.0000	0.0000	0.0000
6	q_6	0	0	$\pi/2$	3.39	0.5	20.2	-27.5	0.0005	0.0028	0.0023	-0.0001	0.0000	0.0000
7	q_7	126	0	0	3.00	-2.0	-19.0	67.0	0.0045	0.0050	0.0060	0.0001	-0.0003	0.0004

(c) Fanuc SR-6iA SCARA manipulator.

#	$\tilde{\theta}_i$	$\tilde{d}_i$	$\tilde{a}_i$	$\tilde{\alpha}_i$	m_i	$r_{x,i}$	$r_{y,i}$	$r_{z,i}$	$I_{xx,i}$	$I_{yy,i}$	$I_{zz,i}$	$I_{xy,i}$	$I_{xz,i}$	$I_{yz,i}$
1	q_1	160	350	0	6.25	209.6	0	44.7	0.0104	0.1355	0.1413	0.0000	0.0056	0.0000
2	q_2	0	300	π	9.49	-166.2	0	-146.4	0.0494	0.1553	0.1336	0.0000	-0.0129	0.0000
3	0	q_3	0	0	0.40	0	0	-175.1	0.0069	0.0069	0.0000	0.0000	0.0000	0.0000
4	q_4	0	0	0	1.10	4.1	-12.4	-16.8	0.0023	0.0004	0.0025	0.0000	0.0000	0.0001

Linear measurements in mm, angles in rad, masses in kg, and moments of inertia in kg m^2. Gravity, in frame 0: $\mathbf{g} = [0 \;\; 0 \;\; -g]^\top$.



(a) KUKA Agilus KR6 R700 (6-DOF). **(b)** KUKA LBR7 (7-DOF). **(c)** Fanuc SR-6iA SCARA (4-DOF).

Figure 3: Frame assignments of the benchmarking manipulators.

The derivation times are listed and are quite quick, even for the 7-DOF KUKA LBR7 robot at 1:28 min. These derivation times are directly related to the maximum linear system size, which grows exponentially with increasing DOF. The number of functions t, before and after pre-filters, is also listed. These numbers illustrate the importance of the pre-filters to the algorithm. While t is a manageable size for the Fanuc and two-link manipulators, for the larger robots the number of search functions is too large for a corresponding $t \times t$ linear system can be solved tractably by most computers. However, after pre-filters, the maximum linear system size is reduced considerably, and PSDM can be applied efficiently.

The table also illustrates the effectiveness of the multiplication-reducing techniques discussed in Section 4.1. For the simple two-link and Fanuc manipulators, the strategy reduces the number of multiplications by 43% and 71%, respectively. However, for the larger manipulators, the reduction is quite substantial, with a 91% reduction for the KR6 manipulator, and 93% for the LBR7. This results in very fast real-time evaluation times for both the forward and inverse dynamic models, and negligible error compared to a direct evaluation using the RNE algorithm.

In the following sections, we will quantify the PSDM algorithm in further detail. Section 5.1 will discuss the derivation times for the algorithms and compare PSDM with a symbolic implementation in Matlab. Section 5.2 discusses the real-time performance of the PSDM model and compares it with other inverse dynamic algorithms. Finally, Section 5.3 will benchmark the model complexity and accuracy trade-off.

5.1. Model Derivation Time

As in Table 2, the PSDM algorithm completed in 5.3 sec for the Agilus robot, 1 : 28 min for the LBR7, and 0.088 sec for the Fanuc SCARA manipulator. To better illustrate the computational complexity of a PSDM derivation for different sizes of kinematic chains, we perform a PSDM derivation on the Agilus and LBR7 robots

Table 2: PSDM derivation results for benchmark manipulators.

		KUKA Agilus	**KUKA LBR7**	**Fanuc SR6iA**	**Two-Link***
	Manipulator DOF	6	7	4	2
	Total PSDM derivation time†	5.3 sec	1:28 min	0.050 sec	0.011 sec
Derivation	No. initial search functions t	$1,306,368$	$10,077,696$	$9,720$	216
	After pre-filters	$17,151$	$88,554$	294	29
	Max linear system size	$3,125$	$15,625$	75	5
	No. base regressor functions p	$6,086$	$21,295$	69	10
	No. base inertial parameters ℓ	36	43	8	4
	No. multiplications‡	$65,745$	$270,954$	435	40
	After optimization	$6,043$ ($\downarrow$ 91%)	$18,387$ ($\downarrow$ 93%)	126 ($\downarrow$ 71%)	23 ($\downarrow$ 43%)
	No. additions‡	$36,720$	$149,352$	300	24
	After optimization	$4,142$ ($\downarrow$ 89%)	$13,981$ ($\downarrow$ 91%)	84 ($\downarrow$ 72%)	12 ($\downarrow$ 50%)
Speed	Mean inverse dynamics error†	$4.1 \cdot 10^{-13}$ Nm	$2.8 \cdot 10^{-13}$ Nm	$4.1 \cdot 10^{-15}$ Nm	$5.1 \cdot 10^{-12}$ Nm
	Mean forward dynamics error†	$1.4 \cdot 10^{-12}$ Nm	$1.6 \cdot 10^{-12}$ Nm	$1.8 \cdot 10^{-14}$ Nm	$1.7 \cdot 10^{-14}$ Nm
	Inverse dynamics eval. time†	2.11 µs	6.49 µs	0.315 µs	0.149 µs
	Forward dynamics eval. time†	2.27 µs	7.28 µs	0.370 µs	0.166 µs

† Derivation time is shown as the median over 100 samples, and function evaluation times are averaged over 10^8 samples. Function evaluation times and errors are evaluated from a set of randomly sampled joint states in the range $\mathbf{q} \in [-\pi, \pi]$ rad, $\dot{\mathbf{q}} \in [-1, 1]$ rad/s, and $\ddot{\mathbf{q}} \in [-10, 10]$ rad/s^2. Error shown is the mean 2-norm of error, compared to the Newton-Euler algorithm, across all samples. Tests done on an Intel i7 CPU@3.2 GHz with 32 GB memory. Real-time functions are procedurally generated in C-code, as described in Section 4.1 and compiled with the MinGW-w64 GCC compiler.
‡ Number of operations are shown for inverse dynamic model – forward dynamic model is similar, with a matrix inversion.
$*$ Evaluated as in previous examples, including simplifying assumptions on inertial parameters. Without these simplifications, results differ slightly, giving $p = 18$, $\ell = 6$.

for the truncated kinematic chain composed of the first i links, for $i = 2, ..., n$. We provide for comparison a custom symbolic routine implemented in Matlab's *Symbolic Toolbox*, which uses the Euler-Lagrange formulation of dynamics to derive the equations of motion of the robot, then split and simplify this model into regressor form. Both algorithms are run on a desktop computer with an Intel i7 CPU @ 3.2 GHz, 32 GB memory. Figure 4 plots the derivation time of both algorithms for each truncated chain, where the dotted lines represent the symbolic results and the PSDM results are displayed with solid lines.

Compared to the symbolic routine, PSDM is very fast. With increasing n, the model derivation times for both algorithms increase at a similar rate, but PSDM in all cases was 2-3 full orders of magnitude faster. We acknowledge that the symbolic routine used in this paper may not be the most efficient or fastest, however the plot does demonstrate that, for small to medium-sized robots, such as those presented in this paper, PSDM can produce results very quickly. PSDM was also found to be more reliable than the symbolic routine, whose performance was unreliable and dependent on the simplification parameters used – sometimes failing to simplify expressions properly and returning unusable results, or crashing due to the large memory used. Illustrating this

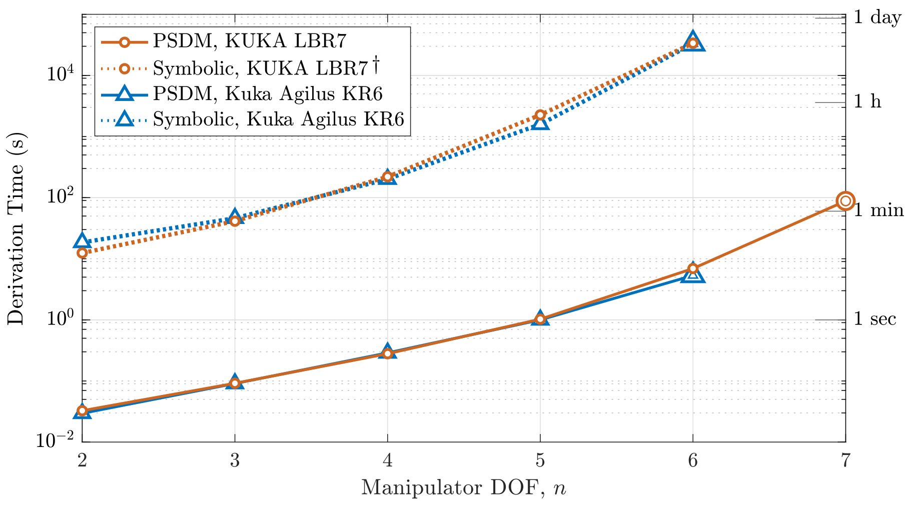


Figure 4: PSDM derivation time of the truncated kinematic chains of the first n links of the Agilus and LBR7 manipulators. The data points for the complete kinematic chains are shown with double-markers (◎ or ▲).
† : Symbolic derivation for the full 7-link chain of the KUKA LBR7 failed after 5 days, due to memory issues.

point, the symbolic derivation for the full LBR7 kinematic chain ($n = 7$) failed entirely due to a lack of memory, in our testing. PSDM, conversely, is very robust, and the derivation time required is primarily bottlenecked by the step of solving the system of linear equations as in (25). Accordingly, its performance is predictable – the maximum size of the solution, t', is known before derivation as in (48), and as such, the estimated derivation time of the model can be estimated beforehand reasonably well.

Figure 4 also highlights the limitations of PSDM. The PSDM derivation time, while quite fast for lower n, scales quite aggressively with increasing n. This effect is due both to the increasing maximum size of linear system t', as in (48), which must be solved during the PSDM derivation, as well as the fact that the solution to a set of linear equations using LU decomposition scales according to a complexity of $O(N^{2.8})$ [33]. For $n < 8$, PSDM is fast, however for larger n, it may not present an attractive or tractable option. In these cases, recursive options, such as the method proposed by Atkeson et al. [17], could be investigated instead.

5.2. Real-Time Performance

Perhaps the most important aspect of a dynamic model is its evaluation speed. To quantify this, we evaluate the inverse dynamics function and average its evaluation time over 10^8 evaluations. No parallel computing is leveraged. Additionally, we perform the same testing on two other algorithms, for comparison.

1. **Symbolic:** The symbolic routine presented previously is used to generate fast real-time code for each manipulator. This code generation is done using the Matlab *Symbolic Toolbox*, which first fully simplifies the model, pre-multiplies constant kinematic properties, and attempts to reduce the number of operations in the evaluation through the use of interim variables (similar to Section 4.1).
2. **Recursive Newton-Euler (RNE), Regressor Form:** We additionally show results using the recursive algorithm presented by Atkeson et al. [17]. This algorithm is a modification to the recursive Newton-Euler algorithm [15], which is linear in the inertial parameters. This algorithm is implemented in the Matlab environment and compiled into C code using Matlab's code generation tools.

The mean evaluation times obtained in these tests are plotted in Figure 5. The results show PSDM to be a very fast option for smaller robots, however with diminishing returns as n increases. Here, the symbolic and PSDM algorithms are non-recursive, which means that their time complexity will increase exponentially as n increases. The recursive Newton-Euler algorithm is recursive and has time complexity $O(n)$. However, it has higher overhead due to the lack of robot-specific simplifications. The non-recursive algorithms can provide aggressive simplifications on smaller robots, resulting in very fast evaluation times. An extreme example of this is the Fanuc SCARA robot, which, in addition to being only 4-DOF, has all of its joints aligned in 3D space. This results in a very simplified dynamic model, which both PSDM and the symbolic routine are able to leverage, but the numerical and non-derived recursive Newton-Euler routine is not. This relationship is better visualized by plotting the performance of the test algorithms for increasing degrees of freedom. As with the derivation times, we use the truncated kinematic chain for $i = 2, ..., n$, for the KUKA LBR7 model, and evaluate the mean inverse dynamic evaluation time as before. These results are shown in Figure 6. In this plot, we see that, for smaller n, the simplifications that the PSDM and Symbolic algorithms offer outweigh their exponential time complexity. However, it is clear that the recursive Newton-Euler algorithm will scale more tractably to

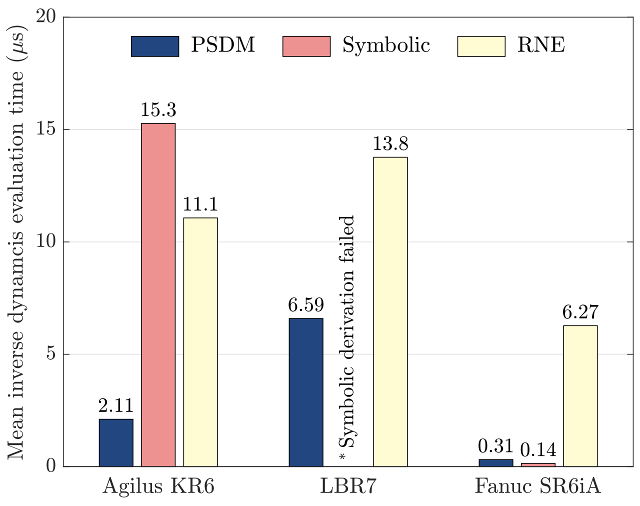


Figure 5: Real-time inverse dynamic model evaluation time for the benchmark manipulators.

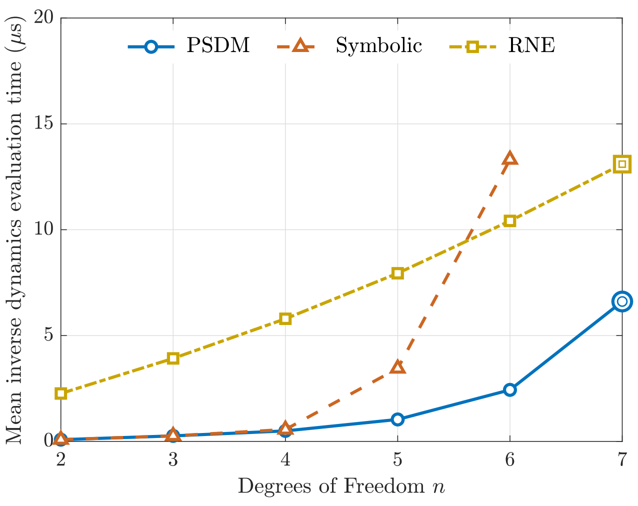


Figure 6: Real-time inverse dynamic model evaluation time of the truncated kinematic chain of a KUKA LBR7 manipulator.

larger n. PSDM then is an attractive option for small to medium-sized kinematic chains, where computation speed is important. For larger chains, however, PSDM becomes unattractive both in terms of derivation time and evaluation time, and other options should be investigated instead.

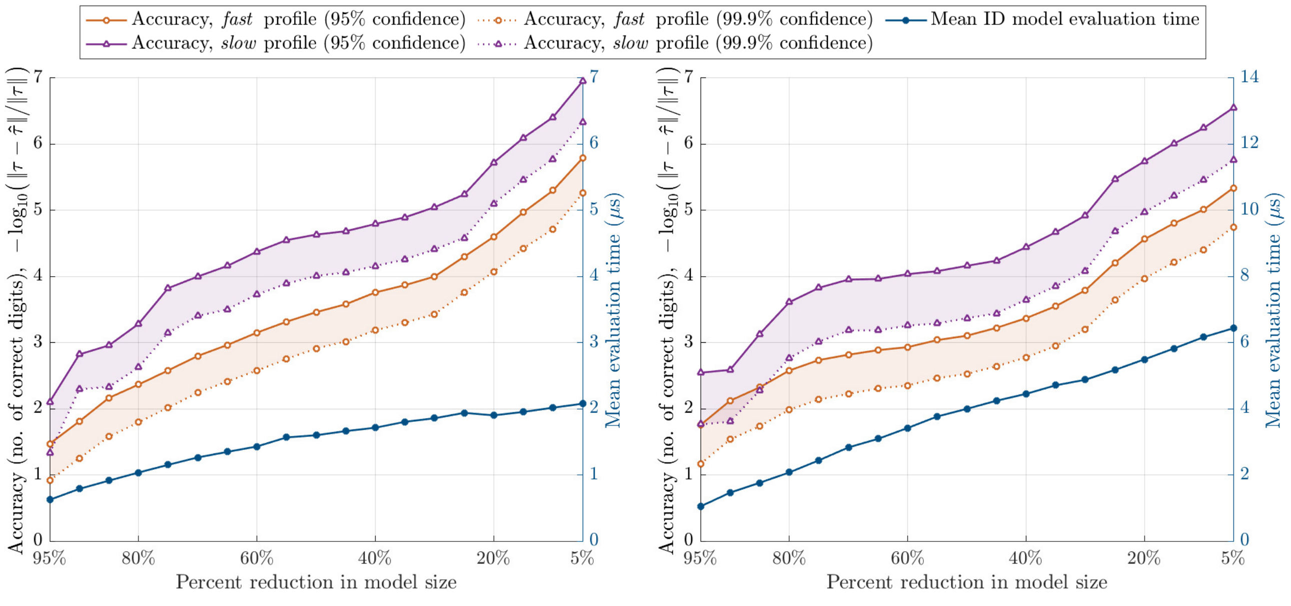
In interpreting these results, we acknowledge that, while the current work has focused on optimizing the PSDM formulation, the comparison algorithms have been implemented "as-is," e.g., without any micro-optimizations. Reasonable attention to speed and efficiency was made, however, it is possible that further optimizations could be made for each. We therefore present these results to show general trends in performance, without making any more absolute claims on the subject. Further research from a computer science, rather than a dynamics perspective, could provide more absolute benchmarking results.

5.3. Model Accuracy and Complexity Reduction

One of the strengths of PSDM lies in its extension to derive non-exact but computationally efficient models, as in Section 4.3. We demonstrate by taking the PSDM models of the Agilus and LBR7 manipulators, and progressively reducing the number of functions in the model, as per the mean of the relative contributions as in (66), over 10^5 samples. We then quantify the evaluation time of each reduced model, as in the previous section. The accuracy of the reduced models is measured using the same "fast" and "slow" speed profiles, as defined in Section 4.3. Model accuracy is quantified with the number

$$-\log_{10}\left(\,\|\boldsymbol{\tau}-\hat{\boldsymbol{\tau}}\|\,/\,\|\boldsymbol{\tau}\|\,\right), \tag{67}$$

which can be interpreted as the *number of correct digits* in the 2-norm of the torque estimate. This accuracy is evaluated over $N = 10^8$ samples and plotted for 95% and 99.9% *confidence* – i.e., the accuracy shown is such that 95% or 99.9% of the sampled values had better or equal performance to the plotted point. For each manipulator, Figure 7 plots the percent reduction in model size along the x-axis, the accuracy on the left y-axis, and overlay the corresponding model evaluation time on the right y-axis. These plots show that, depending on the required accuracy and the speed of the robot, improvements to the model evaluation time are possible. Moreover, these modifications can be performed in an intuitive and numerical manner, using PSDM. We summarize this plot for some key error thresholds in Table 3. For example, in many applications, the torque of a manipulator cannot be controlled with more than 2 or 3 significant digits. Then, depending on the robot speed and the required reliability of the model, a two or three-fold reduction in model complexity can be achieved without affecting the accuracy of the controller. Moreover, in this illustration, our joint sampling was quite general. If joint angles or speeds are known to be further constrained for a given task, even further improvements could be made. In these cases, PSDM offers a procedural and completely numerical means of generating a sufficiently accurate and very fast dynamic model for the manipulator. Moreover, once implemented in code, this algorithm is very accessible – a user can leverage this algorithm with only simple inputs and obtain a usable model in minutes or even seconds, depending on the manipulator size.



(a) KUKA Agilus KR6 manipulator **(b)** KUKA LBR7 manipulator

Figure 7: Complexity reduction trade-off for the Agilus and LBR7 manipulators, with $N = 10^8$ uniformly distributed samples in **(1):** ***Slow* profile** $\dot{\mathbf{q}} \in [-0.1, 0.1]$ rad/s, $\ddot{\mathbf{q}} \in [-1, 1]$ rad/s^2; and **(2):** ***Fast* profile:** $\dot{\mathbf{q}} \in [-1, 1]$ rad/s, $\ddot{\mathbf{q}} \in [-10, 10]$ rad/s^2. Accuracy is plotted such that 95% or 99.9% of the sampled points had equal or better accuracy than the plotted value.

Table 3: Mean evaluation time of the reduced inverse dynamic models for different required accuracy levels.

	KUKA Agilus KR6		KUKA LBR7	
	95% conf.	**99.9% conf.**	**95% conf.**	**99.9% conf.**
2 correct digits (1% error)				
Slow profile	0.62 µs (↓ 70%)	0.79 µs (↓ 63%)	1.05 µs (↓ 84%)	1.76 µs (↓ 73%)
Fast profile	0.91 µs (↓ 57%)	1.15 µs (↓ 45%)	1.47 µs (↓ 77%)	2.44 µs (↓ 62%)
3 correct digits (0.1% error)				
Slow profile	1.04 µs (↓ 51%)	1.15 µs (↓ 45%)	1.76 µs (↓ 73%)	2.44 µs (↓ 62%)
Fast profile	1.43 µs (↓ 32%)	1.66 µs (↓ 21%)	3.77 µs (↓ 42%)	4.88 µs (↓ 25%)

6. Conclusion

This work presented a novel methodology for the derivation of robot dynamics, called *Pseudo-Symbolic Dynamic Modeling*, or PSDM. This method characterized the form of the inverse dynamics of a manipulator, and used this characterization to define a finite set of functions which are guaranteed to span the inverse dynamics function $\boldsymbol{\tau}$ of the manipulator. It was then demonstrated how this set of functions could be reduced into an efficient formulation of the inverse dynamics, using a two-step numerical analysis. This formulation is returned in regressor form, which allows for easy model calibration, or use in adaptive control algorithms. Extensions of the method were presented, allowing for fast real-time code generation, forward dynamic modeling, and a method of sacrificing model accuracy for computational efficiency. The algorithm was implemented in the Matlab environment and benchmarked on several manipulators in terms of derivation time and real-time evaluation speed. The results showed PSDM to be a viable algorithm for small to medium-sized robots ($n < 8$). Moreover, it was demonstrated on 6 and 7-DOF robots that fast dynamic models could be produced procedurally, by reducing the model accuracy within the needs of the application.

PSDM is proposed as an alternative to the current Lagrange and Newton-Euler dynamic modeling methods for small and medium-sized robots, in regressor form, when computational efficiency is needed. It does not require the use of symbolic software and can be implemented in any programming environment using only basic linear algebra operations. PSDM is very flexible in its implementation, and, as it does not rely on potentially proprietary software, very open to general use. Additionally, a full implementation of the algorithm has been coded in the Matlab environment and made available for general use, including several illustrative code snippets implementing examples from this paper.

Supplementary Material

The PSDM algorithm has been implemented in a MATLAB toolbox and is available for download. Up-to-date code is also made available via Github, at github.com/CarletonABL/PSDM. This toolbox allows the PSDM algorithm to be applied to an arbitrary manipulator simply by providing either a DH table `DH` and gravity vector `g`.

```
>> [E, P] = PSDM.deriveModel(DH, g, opt);
```

Implicit simplifications, real time code generation and complexity reduction functions are also provided. Additionally, sample code for many of the manipulators presented in the current work are provided.

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) grants RGPIN-2017-06967, RGPIN-2015-04169, and CRDPJ 514258-17. Support for Steffan Lloyd was additionally provided by the Ontario Graduate Scholarship (OGS) as well as the NSERC CGS-M scholarship.

References

[1] J. He, H. Zheng, F. Gao, H. Zhang, Dynamics and control of a 7-DOF hybrid manipulator for capturing a non-cooperative target in space, Mech. Mach. Theory 140 (2019) 83–103. doi:10.1016/j.mechmachtheory.2019.05.020.

[2] F. Ferraguti, C. Talignani Landi, L. Sabattini, M. Bonfè, C. Fantuzzi, C. Secchi, A variable admittance control strategy for stable physical human–robot interaction, Int. J. Rob. Res. 38 (2019) 747–765. doi:10.1177/0278364919840415.

[3] J. J. E. Slotine, W. Li, Composite adaptive control of robot manipulators, Automatica 25 (1989) 509–519. doi:10.1016/0005-1098(89)90094-0.

[4] J. J. Craig, P. Hsu, S. S. Sastry, Adaptive Control of Mechanical Manipulators, Int. J. Rob. Res. 6 (1987) 16–28. doi:10.1177/027836498700600202.

[5] M. Zeinali, L. Notash, Adaptive sliding mode control with uncertainty estimator for robot manipulators, Mech. Mach. Theory 45 (2010) 80–90. doi:10.1016/j.mechmachtheory.2009.08.003.

[6] M. Galicki, An adaptive non-linear constraint control of mobile manipulators, Mech. Mach. Theory 88 (2015) 63–85. doi:10.1016/j.mechmachtheory.2015.02.001.

[7] M. Tehrani, N. Nariman-zadeh, M. Masoumnezhad, Adaptive fuzzy hybrid unscented/H-infinity filter for state estimation of nonlinear dynamics problems, Trans. Inst. Meas. Control 41 (2019) 1676–1685. doi:10.1177/0142331218787607.

[8] X. F. Zha, Optimal pose trajectory planning for robot manipulators, Mech. Mach. Theory 37 (2002) 1063–1086. doi:10.1016/S0094-114X(02)00053-8.

[9] L. Janson, B. Ichter, M. Pavone, Deterministic sampling-based motion planning: Optimality, complexity, and performance, Int. J. Rob. Res. 37 (2018) 46–61. doi:10.1177/0278364917714338.

[10] S. Ha, S. Coros, A. Alspach, J. Kim, K. Yamane, Computational co-optimization of design parameters and motion trajectories for robotic systems, Int. J. Rob. Res. 37 (2018) 1521–1536. doi:10.1177/0278364918771172.

[11] V. Muralidharan, A. Bose, K. Chatra, S. Bandyopadhyay, Methods for dimensional design of parallel manipulators for optimal dynamic performance over a given safe working zone, Mech. Mach. Theory 147 (2020). doi:10.1016/j.mechmachtheory.2019.103721.

[12] J. J. Uicker, Dynamic Behavior of Spatial Linkages, J. Eng. Ind. (1969) 251–265.

[13] M. E. Kahn, B. Roth, The near-minimum-time control of open-loop articulated kinematic chains, J. Dyn. Syst. Meas. Control. Trans. ASME 93 (1971) 164–172. doi:10.1115/1.3426492.

[14] T. R. Kane, D. A. Levinson, Formulation of Equations of Motion for Complex Spacecraft, J. Guid. Control 3 (1980) 99–112. doi:10.2514/3.55956.

[15] J. Y. S. Luh, M. W. Walker, R. P. C. Paul, On-Line Computational Scheme for Mechanical Manipulators, J. Dyn. Syst. Meas. Control 102 (1980) 69–76. doi:10.1115/1.3149599.

[16] J. M. Hollerbach, A Recursive Lagrangian Formulation of Maniputator Dynamics and a Comparative Study of Dynamics Formulation Complexity, IEEE Trans. Syst. Man. Cybern. SMC-10 (1980) 730–736. doi:10.1109/TSMC.1980.4308393.

[17] C. G. Atkeson, C. H. An, J. M. Hollerbach, Estimation of Inertial Parameters of Manipulator Loads and Links, Int. J. Rob. Res. 5 (1986) 101–119. doi:10.1177/027836498600500306.

[18] W. Khalil, J. F. Kleinfinger, Minimum Operations and Minimum Parameters of the Dynamic Models of Tree Structure Robots, IEEE J. Robot. Autom. 3 (1987) 517–526. doi:10.1109/JRA.1987.1087145.

[19] H. Mayeda, K. Yoshida, K. Osuka, Base parameters of manipulator dynamic models, IEEE Trans. Robot. Autom. 6 (1990) 312–321. doi:10.1109/70.56663.

[20] M. Gautier, Numerical calculation of the base inertial parameters of robots, in: Proceedings., IEEE Int. Conf. Robot. Autom., 4, IEEE Comput. Soc. Press, 1990, pp. 1020–1025. doi:10.1109/ROBOT.1990.126126.

[21] T. Kanade, P. Khosia, N. Tanaka, Real-time control of CMU direct-drive arm II using customized inverse dynamics, in: 23rd IEEE Conf. Decis. Control, volume 53, IEEE, 1984, pp. 1345–1352. doi:10.1109/CDC.1984.272256.

[22] W. Khalil, E. Dombre, Modeling, Identification and Control of Robots, Elsevier, 2004. doi:10.1016/B978-1-903996-66-9.X5000-3.

[23] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, SymPy: symbolic computing in Python, PeerJ Comput. Sci. 3 (2017) e103. doi:10.7717/peerj-cs.103.

[24] M. L. Felis, RBDL: an efficient rigid-body dynamics library using recursive algorithms, Auton. Rob. 41 (2017) 495–511. doi:10.1007/s10514-016-9574-0.

[25] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, N. Mansard, The Pinocchio C++ library, Proc. 2019 IEEE/SICE Int. Symp. Syst. Integr. SII 2019 (2019) 614–619. doi:10.1109/SII.2019.8700380.

[26] R. P. Stanley, Enumerative Combinatorics, Volume 1, 2 ed., Cambridge University Press, New York, NY, 2011.

[27] M. W. Spong, M. Vidyasagar, Robot dynamics and control, John Wiley and Sons, 2008.

[28] G. H. Golub, C. F. V. Loan, Matrix Computations (3rd Ed.), Johns Hopkins University Press, USA, 1996. doi:10.5555/248979.

[29] M. Gu, S. C. Eisenstat, Efficient algorithms for computing a strong rank-revealing QR factorization, SIAM J. Sci. Comput. 17 (1996) 848–869. doi:10.1137/0917055.

[30] J. J. Dongarra, C. B. Moler, J. R. Bunch, G. W. Stewart, LINPACK Users' Guide, Society for Industrial and Applied Mathematics, 1979. doi:10.1137/1.9781611971811.

[31] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, E. Jason Riedy, Error bounds from extra-precise iterative refinement, ACM Trans. Math. Softw. 32 (2006) 325–351. doi:10.1145/1141885.1141894.

[32] C. B. Moler, Iterative Refinement in Floating Point, J. ACM 14 (1967) 316–321. doi:10.1145/321386.321394.

[33] J. R. Bunch, J. E. Hopcroft, Triangular factorization and inversion by fast matrix multiplication, Math. Comput. 28 (1974) 231–231. doi:10.1090/S0025-5718-1974-0331751-8.

[34] H. Olsson, K. Åström, C. Canudas de Wit, M. Gäfvert, P. Lischinsky, Friction Models and Friction Compensation, Eur. J. Control 4 (1998) 176–195. doi:10.1016/S0947-3580(98)70113-X.

[35] P. Vandanjon, M. Gautier, P. Desbats, Identification of robots inertial parameters by means of spectrum analysis, in: Proc. 1995 IEEE Int. Conf. Robot. Autom., volume 3, IEEE, 1995, pp. 3033–3038. doi:10.1109/ROBOT.1995.525715.

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C: The Art of Scientific Computing, 3rd ed., Cambridge University Press, New York, United States, 2007.

[37] B. Armstrong, O. Khatib, J. Burdick, The explicit dynamic model and inertial parameters of the PUMA 560 arm, in: Proceedings. 1986 IEEE Int. Conf. Robot. Autom., volume 3, Institute of Electrical and Electronics Engineers, 1986, pp. 510–518. doi:10.1109/ROBOT.1986.1087644.

[38] A. Izaguirre, R. Paul, Computation of the inertial and gravitational coefficients of the dynamics equations for a robot manipulator with a load, in: Proceedings. 1985 IEEE Int. Conf. Robot. Autom., volume 2, Institute of Electrical and Electronics Engineers, 1992, pp. 1024–1032. doi:10.1109/ROBOT.1985.1087275.

[39] A. Izaguirre, M. Hashimoto, R. P. Paul, V. Hayward, A New Computational Structure For Real-Time Dynamics, Int. J. Rob. Res. 11 (1992) 346–361. doi:10.1177/027836499201100407.

[40] N. Kirćanski, M. Kirćanski, M. Vukobratović, O. Timćenko, An Approach to Development of Real-Time Robot Models, in: Rom. 6, Springer US, Boston, MA, 1987, pp. 603–614. doi:10.1007/978-1-4684-6915-8_62.

[41] Y. R. Stürz, L. M. Affolter, R. S. Smith, Parameter Identification of the KUKA LBR iiwa Robot Including Constraints on Physical Feasibility, IFAC-PapersOnLine 50 (2017) 6863–6868. doi:10.1016/j.ifacol.2017.08.1208.