README – PSDM Matlab Code Package

Steffan Lloyd (steffan.lloyd@carleton.ca)

July 6, 2022

PSDM is a numerical method of deriving the equations of motion of an arbitrary rigid body chain, in regressor form. It is an alternative means of deriving the equations of motion for a rigid, serial kinematic chain. The result is a numerically represented model in a highly organized form, which allows for many symbolic manipulations through numerical methods. This allows for

1. The generation of very fast real time code.
2. Automatic model simplification (to achieve faster evaluation times)
3. Both forward and inverse dynamic modelling in a single derivation and with the same inertial parameter set.

For full details on the PSDM algorithm, please refer to (2) (preprint is given in this repository).

To use PSDM, you need to first derive a PSDM model. This will return an *exponent matrix* **E** and a *reduction page matrix* **P**. These matrices contain the full information to compute the forward and inverse dynamic model, as

$$\tau_i = \mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\, \mathbf{P}_i\, \theta_b, \tag{1}$$

and $\mathbf{y}_p(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ can be calculated from **E** as

$$y_j(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \prod_{i=1}^{5n} \left[\gamma_i(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\right]^{e_{i,j}}, \quad e_{i,j} \in \{0, 1, 2\}, \tag{2}$$

$$\gamma(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) = \begin{bmatrix}\mathbf{q}^\top & \sin(\mathbf{q})^\top & \cos(\mathbf{q})^\top & \dot{\mathbf{q}}^\top & \ddot{\mathbf{q}}^\top\end{bmatrix}^\top, \tag{3}$$

1 Requirements

This codebase requires a Matlab environment of R2018a, or newer. Additionally:

1. The Matlab *Symbolic Toolbox* is used in some of the example live scripts to illustrate some of the derivation results. This toolbox must be installed to run these code snippets.
2. *Matlab Coder* is required to leverage the code-generation capabilities built into this toolbox (see Section 3).
3. The derivation process can also leverage parallel processing, if the Matlab *Parallel Computing Toolbox* is installed. See Section 3 for details on this.

2 Derivation

Derivation of the PSDM model can be accomplished in a single line of code with one of two calling syntaxes.

```
% Can derive model from a DH table and a gravity vector.
[E, P] = PSDM.deriveModel(DH, g);

% Or, can instead give an anonymous function which evaluates the
% inverse dynamics of the manipulator.
[E, P] = PSDM.deriveModel(@inverseDynamicsFunc, jointTypes);
```

2.1 Kinematic Calling Syntax

If the DH / g syntax is used, the DH table should be given as a $n \times 6$ matrix:

$$\mathrm{DH} = \begin{bmatrix} a_1 & \alpha_1 & d_1 & \theta_1 & t_1 & s_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_n & \alpha_n & d_n & \theta_n & t_n & s_n \end{bmatrix} \tag{4}$$

where the DH variables are as defined by (1):

1. a_i is the distance along the x axis from frame o_{i-1} to o_i;

2. α_i is the angular rotation about x_i from o_{i-1} to o_i;
3. d_i is the linear displacement along the z-axis from o_{i-1} to o_i;
4. θ_i the angular rotation about the z-axis from o_{i-1} to o_i;
5. t_i is a number representing the joint type – 0 indicates a revolute joint, 1 indicates a prismatic joint
6. s_i is either -1 or 1, denoting the direction of the joint.

The variables t_i and s_i are combined such that, for each joint, we have

$$d_i^* = d_i + t_i s_i q_i \qquad \theta_i^* = \theta_i + (1 - t_i) s_i q_i \tag{5}$$

The gravity vector is a unit vector which points "upwards" (i.e. against gravity). If omitted, a gravity vector of

$$\mathbf{g} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{6}$$

is assumed, i.e. that the z-axis points directly upwards against gravity.

2.2 Implicit model simplifications

A third parameter $\mathbf{X}$ can also be supplied, which represents the inertial parameters of the robot, in the following form:

$$\mathbf{X} = \begin{bmatrix} m_1 & r_{x,1} & r_{y,1} & r_{z,1} & I_{xx,1} & I_{yy,1} & I_{zz,1} & I_{xy,1} & I_{xz,1} & I_{yz,1} & I_{m,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_n & r_{x,n} & r_{y,n} & r_{z,n} & I_{xx,n} & I_{yy,n} & I_{zz,n} & I_{xy,n} & I_{xz,n} & I_{yz,n} & I_{m,n} \end{bmatrix} \tag{7}$$

where

- m_i is the mass of link i;
- $r_{x,i}$, $r_{y,i}$, $r_{z,i}$ are the (x,y,z) position of the center of gravity of the link;
- $I_{xx,i}$, $I_{yy,i}$, $I_{zz,i}$, $I_{xy,i}$, $I_{xz,i}$, $I_{yz,i}$ are the principle and cross inertia terms of the link, centered at the center of gravity and aligned with the main coordinate system of the link.
- $I_{m,i}$ are the equivalent moment of inertias of the motor armatures and drives. Note, this column may be omitted, if desired, in which case the algorithm will assume the effect of these variables is negligible.

Note, if supplied, the numerical values of this variable are not actually used. However, any parameter which is given as *identically* zero will be implicitely numerically ignored by the algorithm, resulting in a simpler equation. If omitted, the algorithm will simply assume all parameters are nonzero.

2.3 Function calling syntax

If you do not have a DH table of the robot, but instead have access to some dynamic simulation of your manipulator, you can still use PSDM if you can define an anonymous function '@inverseDynamicsFunc' which allows the algorithm to sample the joint torques for any joint states and inertial parameters, as

```
tau = inverseDynamicsFunc(Q, Qd, Qdd, X, g_scale)
```

where `tau`, `Q`, `Qd`, `Qdd` are $n \times N$ matrices (n degrees of freedom, N samples), and $\mathbf{X}$ is an inertia parameter matrix as defined above. Finally, the parameter `g_scale` should be accepted as a scalar which scales gravity. For example, if given `g_scale = 0`, the model should run as if there were no gravity, and if `g_scale = 1`, then the model should run normally (with gravity).

2.4 Additional Options

Additionally, the following name/value pairs can be supplied:

- `tolerance`: A tolerance used in the ID. Any factors below this tolerance (in estimation of the torques) is ignored. Defaut: `1e-10`.
- `verbose`: A verbosity flag. Set to false to suppress output. Default: `true`.
- `gravity_only`: If true, the algorithm will ignore all acceleration and velocity effects. Default: `false`.

3 Using the PSDM Model

3.1 Determining the Parameter Vector

Once the PSDM model has been derived, it can be used as follows. First, the minimal parameter vector $\mathbf{\Theta}_b$ must be determined. A regression of eq. (1) can be used for this, using either numerical or experimental data. To determine $\mathbf{\Theta}_b$ using numerical data, the following function can be used as

```
% Using the DH and g syntax
Theta = PSDM.X2Theta(DH, g, X, opt)
% Or using the functional syntax
Theta = PSDM.X2Theta(inverseDynamicsFunc, linkTypes, X, opt)
```

3.2 Forward and Inverse Dynamics

With a parameter vector defined, the forward and inverse dynamics can be evaluated with the functions

```
tau = PSDM.inverseDyanmics(E, P, Theta, Q, Qd, Qdd)
Qdd = PSDM.forwardDynamics(E, P, Theta, Q, Qd, tau)
```

3.3 Fast Real-Time Code Generation

The functions `PSDM.inverseDynamics` and `PSDM.forwardDynamics` work fine, but are slow. Dedicated fast functions can be automatically generated from a model as

```
PSDM.makeInverseDynamics(filename, E, P, Theta)
PSDM.makeForwardDynamics(filename, E, P, Theta)
```

This will generate a c function defined as per the name given in the `filename` variable, with many optimizations done. This code can also be compiled into a MEX file to be used in Matlab. These generated functions are highly optimized and are very fast.

3.4 Complexity Reduction

This codebase also includes tools to reduce the complexity of the model without significantly reducing model quality. The usage of this is summarized in the help text of the function

```
[Eh, Ph] = PSDM.reduceModelComplexity(DH, g, X, E, P, options)
% Or, using the function syntax
[Eh, Ph] = PSDM.reduceModelComplexity(inverseDynamicsFunc, ...
    linkTypes, E, P, options)
```

4 Making PSDM Code

Many of the functions in PSDM run fairly slow in interpreted Matlab code. These can be sped up by compiling them into MEX files, if the user has Matlab Coder installed. To do this, run the command:

```
PSDM.make();
```

This will compile many of the PSDM functions into mex files. Then, to tell the package to "use" the mex files, you need to modify the file `+PSDM/config.m` such that the `use_mex` parameter is `true`.

Additionally, if you have the Parallel Computing Toolbox installed, you can set the `use_par` parameter to `true`, in order to enable the derivation scripts to use parallel processing to further speed up the code.

5 More Help

All the functions in this toolbox are written in a way to integrate with Matlab's `help` function. To get additional details on the calling syntax or options of any of the functions, you can simply type.

```
help PSDM.functionName
```

6 Examples

Several examples are given for common manipulators, including

1. Two-link planar manipulator;
2. SCARA robot;
3. KUKA KR6 spherical wrist 6-DOF manipulator.

Photos and numbers are given in the appropriately named scripts in the `examples` folder.

7 MECC 2022 Supplementary Material

A paper describing the usage of PSDM on a Denso 6556W manipulator has been accepted for publication in the MECC 2022 conference. Supplementary material for this publication is provided in the directory `supplementary_material_MECC2022`.

8 Crediting and Contact

This software is developed by Steffan Lloyd (email steffan.lloyd@carleton.ca). The code is offered "as is" and can be used at the risk of the user.

If you use our work in a project, we ask that you credit us appropriately (through a citation or otherwise). The work should be cited as (2)

```
[1] S. Lloyd, R. Irani, and M. Ahmadi, ‘‘A numeric derivation for fast
regressive modeling of manipulator dynamics,’’ Mech.  Mach.  Theory, vol.
156, p.  104149, Feb.  2021.
```

This work can be accessed via its doi at `10.1016/j.mechmachtheory.2020.104149`. A preprint version of this work is included in this repository, see `SLloydEtAl2021_PSDM.pdf`.

References

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

[2] S. Lloyd, R. Irani, and M. Ahmadi, “A numeric derivation for fast regressive modeling of manipulator dynamics,” Mech. Mach. Theory, vol. 156, p. 104149, Feb. 2021.