Model based code generation for kinematics and dynamics computations in robot controllers

Marco Frigerio, Jonas Buchli and Darwin G. Caldwell

Department of Advanced Robotics, Istituto Italiano di Tecnologia (IIT), via Morego, 30, 16163 Genova
{marco.frigerio, jonas.buchli, darwin.caldwell} @iit.it

Abstract— **This paper gives an overview on our current research on models and Domain Specific Languages (DSLs) for robotics software development. We target the set of kinematics and dynamics algorithms which are critical for simulation and control (e.g. forward kinematics or inverse dynamics), but whose manual implementation is error prone and time consuming. Our approach relies on model based code generation to address the apparently conflicting requirements arising from the constraints of real time robot controllers on one hand, and the ease of use and the flexibility desirable by the end user on the other hand. Evidences from our experience and numerical experiments demonstrate the effectiveness of this approach.**

I. INTRODUCTION AND RELATED WORK

Model based controllers for articulated robots, such as operational space controllers [1] and impedance controllers [2], heavily rely on kinematics and dynamics algorithms and expressions. Despite an established theoretical understanding of such algorithms, sound implementations demand a lot of resources, making the development of new experimentations with robots harder.
The difficulties can arise from the lack of established software *models* and thus the lack of *reusable* components, in addition to the inherent complexity of kinematics/dynamics. The implementation becomes even more challenging when real time constraints and limited hardware resources demand fast and efficient code.

To address these issues we designed some simple yet general domain models of the relevant aspects of the kinematics and dynamics of a robot, and then built DSLs on top of them [3], with the final purpose of automatically generating code. We primarily target the algebraic expressions typical of different control approaches (e.g. operational space control, Jacobian-transpose force control) and rigid body dynamics algorithms.
Our approach exhibits several desirable features which include:

- ease of use: the user is required to deal only with high level information;
- robustness: an automated code generation process is repeatable and cannot introduce occasional mistakes;
- flexibility: domain models make the software more general and enable extensibility;
- efficiency: the generated implementations can be optimized to address the speed and efficiency constraints of real time robot controllers.

In principle DSLs allow to generate code of any language, but for this work we mainly focus on C/C++ (using the algebra library Eigen [4]) because of the need to run hard real time control loops at high frequencies (e.g. 1Khz). We develop the DSLs with the Xtext workbench for Eclipse [5], using the Java and the Xtend2 languages [6]. In addition, we rely on the Maxima symbolic computation engine [7], to simplify algebraic expressions and thus generate optimized code.

Generation of code and equations is a feature available in some commercial software packages, as for instance SD/FAST [8] and Robotran [9]. SL is a mature package [10] which is used in several research labs – including ours – for simulations and control; we will use it for some comparisons with our code. We will also show some comparisons with the Stanford Whole Body Control (`S-wbc`) software, which explicitly targets the implementation of the operational space formulation [11]. For further details about the kinematic model and rigid body dynamics algorithms please refer to the work of Featherstone [12].
Our contribution lies in gathering different points: the use of domain models according to an established best practice in software design (in a field like robotics, that on the other hand seems to partially lack of a principled development process [13]); the use of the technological opportunity given by DSLs to realize our purposes; the efficiency of the resulting implementations, which meet the requirements of controllers of real robots.

II. MODELS AND LANGUAGES

Figure 1 shows an UML class diagram representing kinematic trees. The model is simple but general, and can be applied to almost any robot made by rigid links (we are not handling kinematic loop though).

We designed a DSL based on this domain model so that its instance documents are specific robot descriptions (an example in Figure 2). The effort required to design the grammar of the language was relatively limited and subject to a confident understanding of the domain.
With this DSL the user only needs to produce an high level description of the robot, which however fully specifies the physics of the system according to the rigid body dynamics model.
Then, such information can be transformed automatically

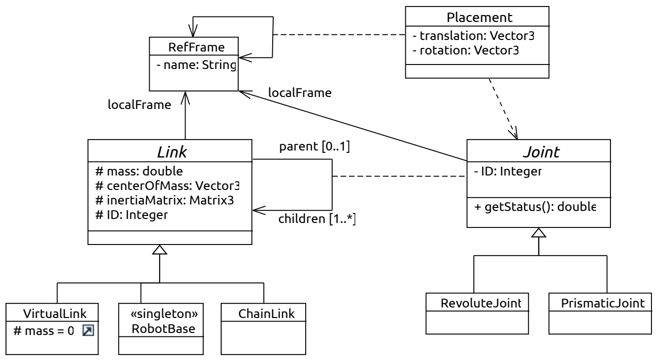


Fig. 1. The kinematic tree (meta)model as an UML class diagram. It can represent a tree-like layout of rigid bodies, capturing the role of joints as the connection between any pair of parent/child bodies, and emphasizing their association with reference frames. The convention about the placement of these frames and numerical data about the relative pose of two successive ones (the `Placement` class) provide the full geometry information about the robot.

into executable code, relieving the user from manual development: for instance, rigid body dynamics algorithms like the Newton-Euler inverse dynamics are parametrized on such robot models, therefore robot-specific, optimized implementations can be generated.

```
Robot Fancy {
RobotBase FancyBase {
    inertia_params{...}
    children { link1 via jA }
}

link link1 {
    id = 1
    inertia_params {
        mass = 1.0
        CoM = (0.5, .0, .0)
        Ix=0.0025  Iy=0.084  Iz=0.084
          Ixy=0.0  Ixz=0.0  Iyz=0.0
    }
    children { link2 via jB }
}
                     ...
```

```
r_joint jA {
    id = 1
    ref_frame {
        translation = (0.0, 0.0, 0.0)
        rotation = (0.0, 0.0, 0.0)
    }
}
p_joint jB {
    id = 2
    ref_frame {
        translation = (1.0, 0.0, 0.0)
        rotation = (-PI/2.0, 0.0, 0.0)
    }
}
                 ...
```

Fig. 2. A little excerpt from a document of the kinematic DSL describing a fictitious 5 DoF robot. The text shows the fixed base, the first link and the first two joints.

Another possible exploitation of the robot model pertains kinematics, and in particular the coordinate transformations and the geometric Jacobians. These matrices typically suffer from ambiguities that make their manual development tricky and time consuming. An effective software model of these objects exposes the properties that allow to disambiguate the structure of the expressions (e.g. whether a rotation matrix is supposed to multiply a vector in the original frame or in the rotated one), enabling, for instance, consistent code generation. See Figure 3.

Part of the diagram in Fig.3 served as the basis for another standalone DSL, whose documents contain an abstract specification of some transformation matrices. This abstract description allows to generate different matrices like homogeneous transforms or 6D motion vector transforms.
We take advantage of this language by translating the geometric data about the placement of robot frames to a user defined list of abstract transforms. Then, these can in turn

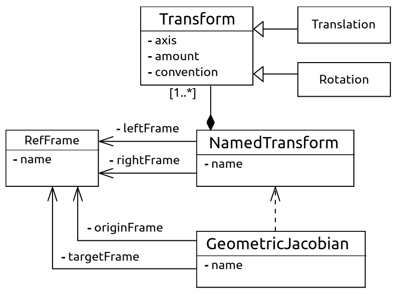


Fig. 3. The model of coordinate transformations and Jacobians as an UML class diagram. Transforms are composition of basic rotations and translations; named transforms describe the relative pose of two known frames; `left` and `right` refer to the position of frames B and A in the notation ${}_BX_A$. Jacobians on the other hand are fully determined by a pair of frames, as they can be computed from the transformations related to such frames.

be transformed into executable code.
Jacobians, on the other hand, do not add further concepts to the model, and their generation happens directly from the kinematic model (though they depend on the transforms of direct kinematics).

Currently, our DSLs allow to generate the following elements:

- Robot model description: URDF XML file (used in ROS), Matlab model (compatible with Featherstone's code), SL model.
- Dynamics (C++): the Newton-Euler algorithm for inverse dynamics, the composite-rigid-body algorithm for the joint space inertia matrix M, the L^TL factorization of M.
- Kinematics (C++, Maxima): coordinate transforms and Jacobians

Figure 4 gives a rough idea on how the C++ code might look like.

```
transforms6D::fr_link2_X_fr_link1(q);
link2_v = (transforms6D::fr_link2_X_fr_link1 * link1_v);
link2_v(5) += qd(1);
Utils::fillAsMotionCrossProductMx(link2_v, spareMx);
link2_a = (transforms6D::fr_link2_X_fr_link1 * link1_a) +
                                  (spareMx.col(5) * qd(1));
link2_a(5) += qdd(1);

link2_f = link2_Imx * link2_a +
                 (-spareMx.transpose() * link2_Imx * link2_v);
```

Fig. 4. A little section of the generated C++ code for the first pass of the Newton-Euler inverse dynamics. The first line is computing the transform ${}_{link2}X_{link1}$ according to the new joint status `q` (i.e. $\boldsymbol{q}$); similarly, `qd` and `qdd` represent $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$. `v`, `a` and `f` stand for velocity, acceleration and force (of the link).

III. Experimental results

In this last section we will show some performance evaluations of the generated code (all tests were executed on a Intel(R) Core(TM)2 Duo CPU, P8700 @ 2.53GHz). We obviously performed other tests to assess the numerical correctness, generally by comparing the output with other

established software.
Our approach has proven to be very convenient and flexible, in that given a new robot model is very easy to get working implementations of diverse kinematics/dynamics computations.

Figure 5 shows a comparison of the execution time between our code and SL, for the computation of the Newton-Euler inverse dynamics. SL generates a highly optimized C code implementation, and in our experience its performance can very well be considered as a reference. As can be seen

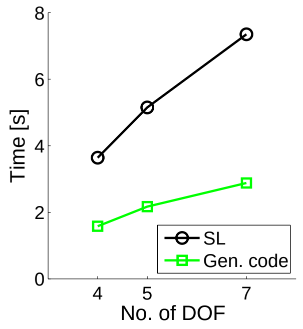
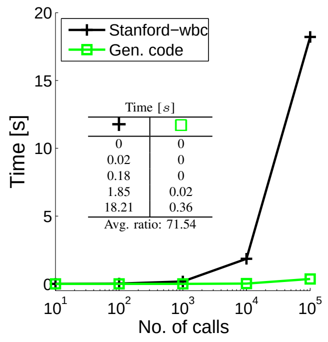


Fig. 5. Performance comparison with the SL software about the Newton–Euler algorithm for inverse dynamics. The plot shows the cumulative execution time for 10^6 calls of the function $\boldsymbol{\tau} = f(\ddot{\boldsymbol{q}},\ \boldsymbol{q}, \dot{\boldsymbol{q}})$ as a function of the number of degrees of freedom of three robot models.

from the plot, the execution times of the two implementation basically have the same order of magnitude. We achieved a performance similar to SL without sacrificing the usability and the maintainability of the modeling/generation process.

Figure 6 shows instead a comparison with the S-wbc software, about the computation of the base-to-end-effector Jacobian and the inverse of the joint space inertia matrix, for a 5 DoF robot. These quantities are the factors of the inertia weighted pseudo inverse of the Jacobian, and then of its null space projector, an expression that appears in the operational space formulation.

The graph clearly shows a better performance for the code



Time $[s]$

+	□
0	0
0.02	0
0.18	0
1.85	0.02
18.21	0.36

Avg. ratio: 71.54

Fig. 6. Performance comparison with the S-wbc for the computation of the inverse of the joint space inertia matrix M^{-1} and the end-effector Jacobian J, for a 5 DoF robot. The x axis represent the number of calls to the update functions $M(\boldsymbol{q})^{-1}$ and $J(\boldsymbol{q})$, the y axis the total execution time in seconds. The same random joint state vector $\boldsymbol{q}$ is used for each call on both implementations.

generated with our approach, due to the efficient algorithms we are employing to compute M^{-1} (the composite-rigid-body algorithm and the $L^T L$ factorization) and J. The computation of M makes use of several coordinate transforms, which are also efficient thanks to the symbolic simplifications taking place in the generator. The same thing applies for J.

IV. Conclusions

In this paper we have shown an approach based on simple domain models and Domain Specific Languages to support the development of robotics software by automating the implementation of kinematics and dynamics computations.
The use of shared models among the components and the code generation improve the ease of use and the flexibility of our tools. Experimental results also demonstrate the compatibility with applications with hard real time constraints, such as fast controller of real robots.

Acknowledgements

This research has been funded by the Fondazione Istituto Italiano di Tecnologia.
The authors would like to thank the developers and the maintainers of Xtext, Xtend, Maxima, Eigen for having created these softwares and having released them to the community.

Publications

The work described in this extended abstract is detailed in:

- M. Frigerio, J. Buchli and D. G. Caldwell, A Domain Specific Language for kinematic models and fast implementations of robot dynamics algorithms, in *2nd International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLRob)*, 2011.
- M. Frigerio, J. Buchli and D. G. Caldwell, Code Generation of Algebraic Quantities for Robot Controllers, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012 *[submitted for review]*.

References

[1] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.
[2] N. Hogan, “Impedance control: An approach to manipulation: Part II – Implementation,” *ASME, Transactions, Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 8–16, 1985.
[3] M. Fowler, *Domain-Specific Languages*. Addison-Wesley, 2010.
[4] G. Guennebaud, B. Jacob, *et al.* The eigen library v3. [Online]. Available: http://eigen.tuxfamily.org
[5] M. Eysholdt and H. Behrens, “Xtext: implement your language faster than the quick and dirty way,” in *ACM international conference companion on Object oriented programming systems languages and applications companion*, ser. SPLASH ’10. New York, NY, USA: ACM, 2010, pp. 307–309.
[6] The xtend language. [Online]. Available: http://www.xtend-lang.org/
[7] Maxima. (2011) Maxima, a computer algebra system. version 5.25.1. [Online]. Available: http://maxima.sourceforge.net/
[8] M. Sherman and D. Rosenthal. Sd/fast. [Online]. Available: http://www.sdfast.com/
[9] Robotran. [Online]. Available: http://www.robotran.be/
[10] S. Schaal, “The sl simulation and real-time control software package,” CLMC lab, University of Southern California, Tech. Rep., 2009.
[11] R. Philippsen, L. Sentis, and O. Khatib, “An open source extensible software package to create whole-body compliant skills in personal mobile manipulators,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2011, pp. 1036 –1041.
[12] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2008.
[13] R. Bischoff, T. Guhl, E. Prassler, W. Nowak, G. Kraetzschmar, H. Bruyninckx, P. Soetens, M. Haegele, A. Pott, P. Breedveld, J. Broenink, D. Brugali, and N. Tomatis, “BRICS – best practice in robotics,” in *IFR International Symposium on Robotics (ISR)*, 2010.