ModelRob: A Simulink Library for Model-Based Development of Robot Manipulators

Indranil Saha and Natarajan Shankar

Abstract— **Robot manipulators are widely used in many industrial automation applications. A robot manipulator moves the end-effector to the configuration instructed by the user. The user input from a master unit is transformed into the desired configuration through forward kinematics. This configuration is communicated to the robot controller, which employs inverse kinematics to transform the configuration into joint angles. The control algorithm is implemented as software and embedded into the robot controller. The software is typically written in traditional programming languages like C or C++. We introduce a Simulink Library *ModelRob* that provides basic building blocks to model kinematics of a robot manipulator. Availability of such a library enables Model-Based Development (MBD) of robot manipulator software, where the manipulator controller can be modeled using *ModelRob* library blocks, and production code can be automatically generated using existing code generators for Simulink. We enlist the existing tools that can be useful in the verification and validation stage of the MBD process, and outline the need for tool-support for verification activities specific to building robust robot manipulator software. Using *ModelRob* library we have modeled Cartesian space motion controller of a robot manipulator in Simulink and successfully generated C code from the model.**

I. Introduction

Model-based development approaches are increasingly popular in safety-critical domains such as the automotive and aerospace industries. Intuitively, model-based development means that the main activity in the development process is designing a model of the system from the informal specification. The target code is then generated from the model using an automatic code generator. Model-based development has two key benefits over traditional software engineering processes. First, the engineers in a particular domain do not need to know the syntax of the target programming language. They can design the model of the system using block diagrams available in modeling languages. The target code is automatically generated from the model. Second, the models are easier to maintain and understand than the code.

Robot manipulators [1], [2] find applications in many mission-critical systems, for example, spacecraft [3], [4] and life-critical systems, for example, tele-surgical robots [5], [6], [7]. The advantages of applying model-based development in safety-critical automotive and avionics applications also apply to the development of robot manipulator software for mission-critical and life-critical applications.

Simulink [8] is a high-level model design tool which is very popular in many industrial application domains. Simulink provides a wide range of library blocks, for example, Arithmetic blocks, Logic and Bit Operation blocks, Signal Routing blocks, to name a few. By using subsystems, it is possible to build a model of a large system in a hierarchical manner. There are commercial code generators available for Simulink, including *Real-Time Workshop* from MathWorks [9] and *TargetLink* from dSpace [10]. Moreover, a number of verification and testing tools have been developed in the last decade for Simulink based development of embedded software.

The centerpiece of a model-based development process is a modeling tool associated with a library specific to the domain of interest. In this paper our objective is to show the feasibility of model-based development of robot software using Simulink. Though we focus on Simulink, the approach presented in this paper is applicable to any tool that supports high-level modeling of basic mathematical operations required to design a robotic application. *Robotics Toolbox* [11], a Simulink library, enables us to model and simulate behaviors of robot manipulators in Simulink. However, *Robotics Toolbox* is not suitable for model-based development, as the Simulink blocks are written as Matlab S-functions [12], rather than being developed using basic Simulink library blocks. As there is no available code generator that can generate code from Matlab S-functions, *Robotics Toolbox* is not useful for generating production code automatically. We have developed *ModelRob*, a library for Cartesian space motion control for robot manipulators with revolute axes. We believe that the availability of such a library will drive the model-based development of robot manipulators in the industrial level. Our library can be used to model a robot manipulator, simulate it, and generate controller code from the model. Using the library, we have successfully modeled Cartesian space motion controller of a robot manipulator and generated C code for it. Moreover, we can also generate fixed-point model from the floating-point model by selecting proper fixed-point data-types for the basic blocks in the model. This enables us to develop controller code that can run on fixed-point processors. The library and the example models are available for academic use at: `http://www.csl.sri.com/~shankar/ModelRob.tar`.

There are a few other libraries available for mathematical computations for robot kinematics [13], [14], [15]. These libraries are written in high-level programming languages, like C++. Recently Geisinger et al. [16], proposed a software architecture for model-based programming of robot systems. Their architecture is based on *EasyLab* [17], a model-based development tool for software components of embedded systems. Broenink et al. [18] proposes a model-driven design methodology based on the co-simulation of the discrete-

This research was supported by NSF Grant CSR-EHCS(CPS)-0834810 and NASA Cooperative Agreement NNX08AY53A.

I. Saha is with the Computer Science department, University of California, Los Angeles, CA 90095, USA `indranil@cs.ucla.edu`

N. Shankar is with the Computer Science Laboratory, SRI, Menlo Park, CA 94250, USA `shankar@csl.sri.com`

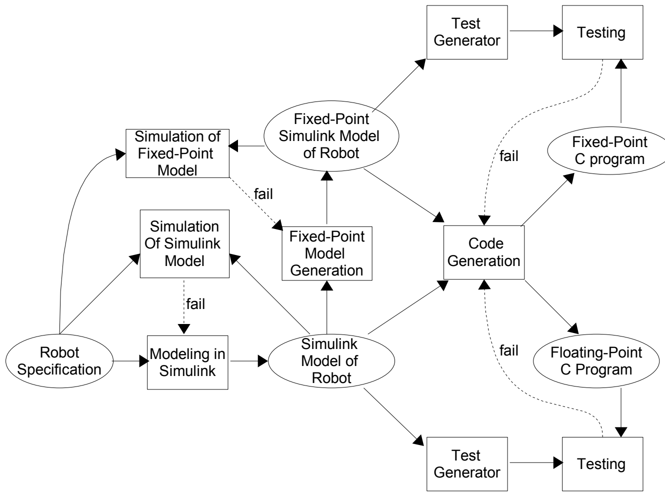


Fig. 1. The model-based development process for robot manipulators

event controller model and the continuous-time model of robot mechanism. Their model of controller is built using a graphical tool that is capable of simulating the model and generating code from the model. However, we champion the use of Simulink as the core of the model-based development process, since it is supported by mature automatic code generators. Also, a number of verification and testing tools have been developed in the last decade for Simulink based development of embedded software. We can leverage these tools for the model-based development of robot manipulator software.

Schlegel et al. [19] propose the use of UML to model interaction of different loosely coupled components of a robotic software. In this paper, our focus is modeling of the controller component of a robot, generating code automatically from the model and providing verification support in this process. Our model-based development methodology can be seamlessly incorporated in the model-driven design methodology proposed by Schlegel et al. [19]. The benefits of combining the low level Simulink model for a component with the high level UML model of the complete system have been studied recently [20], [21] and our thesis is that similar approach in developing complex robotic software will be very effective.

The rest of the paper is organized as follows. In Section II, we provide some background on robot manipulators. In Section III, we introduce the *ModelRob* library and describe the proposed model-based development process for robot manipulator software. In Section IV, we illustrate different aspects of the model-based development process on an example of Cartesian space motion control of a robot manipulator. Section V concludes the paper and enumerates possible directions of future research in this area.

II. Robot Manipulators

A Robot manipulator can be represented as a chain of rigid bodies, called *links* connected by *joints*. Each joint has either a translational or a rotational degree of freedom. The first kind is called a *prismatic* joint and the second is called a *revolute* joint. One end of the manipulator chain is constrained to the base, the other end which is free is connected to an *end-effector*. A link is specified by two parameters: the *link length* and the *link twist*. These two parameters define the relative location of the two axes in space. A joint is specified by two parameters: the *link offset* and the *joint angle*. The link offset is the distance between two consecutive links along the axis of the joint. The joint angle is the rotation of one link with respect to the next about the joint axis. These four parameters together are called *Denavit-Hartenberg (DH) parameters* [22]. The end-effector's position can be controlled using either the joint angle or the link offset. For a revolute axis, the joint angles are the control variables and the link offsets are constant, whereas for a prismatic joint, link offsets are controllable, and joint angles are constant. We may describe the position of the joints of the manipulator with respect to the coordinate frame associated with the base, called the *base coordinate frame* or the coordinate frame associated with the end-effector, called the *end-effector coordinate frame*.

Two very basic problems related to the motion of the manipulator are *forward kinematics* and *inverse kinematics*. In *forward kinematics* the position and orientation of the end-effector is computed from a given set of joint angles. On the other hand, the problem of calculating all possible sets of joint angles from a given position and orientation of the end-effector is referred as the *inverse kinematics* problem. In forward and inverse kinematics we statically deal with position of the end-effector and the joint angles of the manipulator. However, we are often interested in analyzing the motion of the end-effector. The velocity analysis is performed using a matrix quantity called the *Jacobian*, which transforms the velocities in the joint space to the velocities in the Cartesian space.

There are many other interesting topics related to robot manipulators, such as trajectory generation, position control, force control, and so on. Interested readers are referred to the textbook written by Craig [1].

III. The Model-Based Development of Robot Manipulators

The proposed process for model-based development of a robot manipulator is shown in Figure 1. In the figure, a rectangle is used to represent a process step, and an oval to represent an input or output entity of a process. The first step in the process is to create a Simulink model from the specification of the robot system. The Simulink model is then verified to ensure that the behavior of the model is in congruence with the specification. Once the Simulink model complies with all high-level requirements in the specification, target code is generated automatically from the model. Moreover, test cases are generated from the model using some coverage criteria. The test cases are used to test the generated code to ensure that no error has been introduced during the compilation process. If the target processor is of fixed-point type, then the Simulink model is converted into a fixed-point model and the model

is verified against the performance-related requirements in the specification. Once the fixed-point model is verified, it is used to generate fixed-point code. Fixed-point code is tested using the test cases generated from the fixed-point Simulink model. In the following subsections, we detail the key steps in the process.

A. Modeling of a Robot Manipulator

Since the key step in the MBD process is modeling the robot manipulator, we develop a Simulink library that contains blocks implementing the key functionalities. We call our library the *ModelRob Library*. Unlike the *Robotics Toolbox* [11], the Simulink blocks of *ModelRob Library* have been developed using Simulink's native library blocks. Our library currently contains Simulink blocks that can be used to model a manipulator that consists of revolute axes. The Simulink blocks are parametric in the DH parameters for the links, the values of which for a particular robot can be provided through a Matlab script. Currently our library contains the following blocks (the names of the blocks are the same as in the *Robotics Toolbox*):

- **jacob0** Compute Jacobian in base coordinate frame
- **jacobn** Compute Jacobian in end-effector coordinate frame
- **ijacob** Compute inverse Jacobian
- **fkine** Compute forward kinematics
- **tr2diff** Compute differential motion vector from homogeneous transform
- **xyz2T** Compute homogeneous transform from Cartesian coordinate
- **T2xyz** Compute Cartesian coordinate from homogeneous transform
- **rpy2T** Compute homogeneous transform from Roll/Pitch/Yaw angles
- **T2rpy** Compute Roll/Pitch/Yaw angles from homogeneous transform

The above blocks are sufficient for Cartesian space motion control of the end-effector of a robot manipulator.

B. Code Generation

The main advantage of using native Simulink blocks to implement the blocks in *ModelRob* Library is that we can use the code generators for Simulink models to generate code automatically from the manipulator model. There are two commercial code generators From Simulink models: *Real-Time Workshop* (RTW) from MathWorks [9] and *TargetLink* from dSpace [10]. Real-Time Workshop is widely used in the automotive and avionics industries. It can generate C/C++ code for different hardware platforms, and can perform various code optimizations that enhance the performance of the object code.

In many embedded applications, fixed-point processors are preferred to floating-point processors due to their lower prices and enhanced performance. Though fixed-point implementation causes loss of precision in the software output, the loss of precision may be acceptable for many applications. We can generate fixed-point C code from the Simulink model by first converting the Simulink model to fixed-point Simulink model using Simulink Fixed-Point tool [23], and then using RTW code generator on the converted model.

C. Verification and Validation

The model-based development process brings new challenges for verification and validation. The first verification objective is to make sure that the robot model complies with high-level requirements of the specification. Simulation is mostly used to have confidence on the Simulink models. Research has been carried out to increase the simulation coverage of Simulink models [24]. While simulation is effective at finding bugs in the model, it cannot guarantee correctness. There are a few tools available that deal with verification of safety properties of Simulink models[25], [26], [27], [28]. However, the nonlinear nature of the robot manipulator model brings new challenges to verification of properties that are not addressed by any of these tools.

For a robot manipulator the following two properties mainly are of interest:

Prop 1 While moving the end-effector, no part of the robot manipulator should go inside an unsafe region.

Prop 2 Eventually the end-effector follows the desired trajectory within a certain specified error bound.

Sharygina et al. [29] formulated 37 properties for a NASA robot controller including various forms of the above two general properties, and successfully verified 22 of them. Their robot controller is implemented in C++. However, in the model-based development process we need to verify the model, as target code can be automatically generated from the verified model.

Once we establish that the Simulink model complies with the specification, the model can be treated as the formal specification, and test cases can be generated from the model to test the generated code. Though the code is automatically generated, it should still be tested to ensure that errors have not been introduced in the selection of code generation or compiler options, or in the naming and inclusion of library files [30].

There are many tools that generate test cases from Simulink models, for example, HiLite [30], Reactis [31] and T-VEC [32]. As exhaustive testing is not practical, test generation is normally done using some coverage criteria. For testing of safety-critical systems, MC/DC coverage criteria [33] is mandated by certification regulations such as DO-178B [34]. As robot manipulators are also used in life-critical and mission critical applications, the same rigor in testing should be in place.

D. Implementation for Fixed-point Processors

If the target processor for the robot manipulator software is of fixed-point type, we need to generate a fixed-point Simulink model from the original Simulink model, and then generate fixed-point code from it. However, due to precision loss in the fixed-point implementation, the desired trajectory followed by the end-effector may display significantly more error than its floating-point counterpart, and the error margin decreases with increasing number of bits in the fixed-point processor. The robot specification should include the error

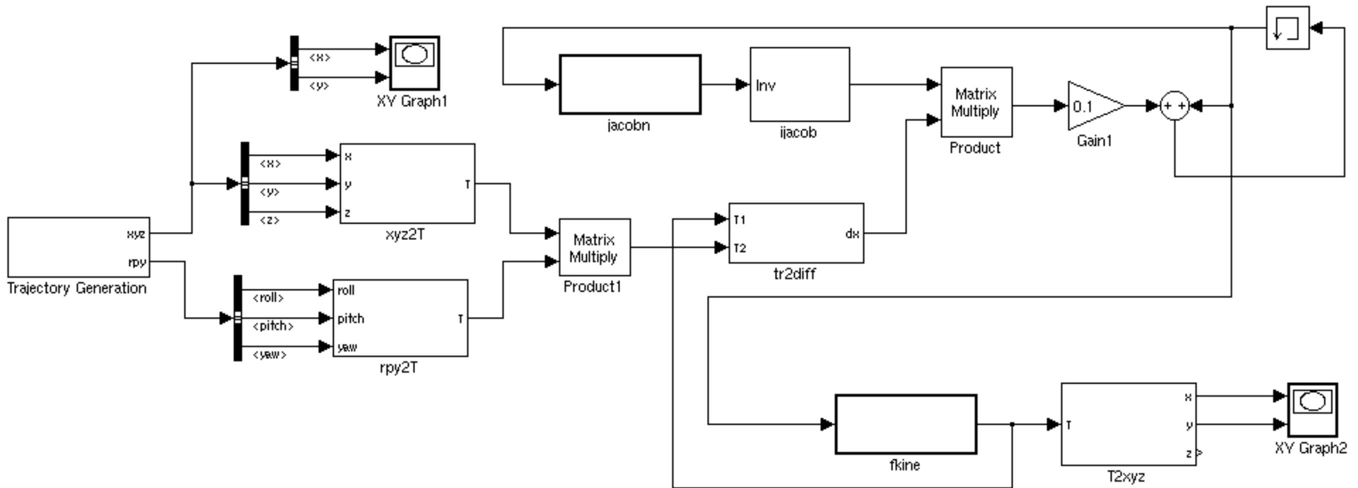


Fig. 2. Simulink model for Cartesian space motion control of a robot manipulator

tolerance in the trajectory, and this should drive the decision of choosing a particular fixed-point implementation (number of bits in the processor). However, starting from such a requirement it is difficult to choose a particular number of bits to represent the variables so that the requirement is satisfied. Hence the bit-size is chosen in an iterative manner. Starting from an implementation with a low number of bits, one can attempt to verify the error tolerance property for the implementation. If the property is not satisfied, one need to go for an implementation with a larger number of bits.

IV. CASE STUDY

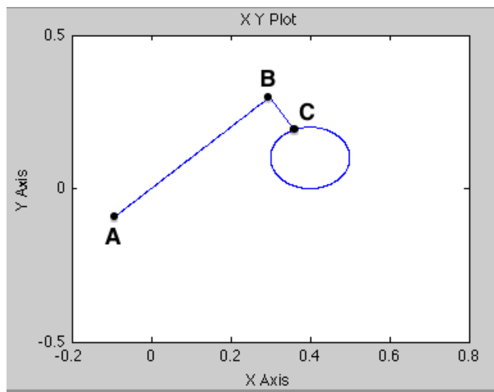
As a case study we have undertaken Cartesian space motion control of a robot manipulator. The objective here is that the end-effector of the manipulator should follow a desired trajectory given as the input. This can be achieved in two ways. Firstly, to perform the control in the joint space, the Cartesian space demand can be resolved to joint space through inverse kinematics. Secondly, the error in Cartesian space can be used to resolve the error in the joint space via the inverse Jacobian. We adopt the second approach and model the controller using *ModelRob Library* blocks. The Simulink model is shown in Figure 2.

We have experimented with a few input trajectories, for example, a constant point, a step function, and a circular trajectory. In Figure 3, we show how the end-effector tracks an arbitrary trajectory. Figure 3(a) and Figure 3(b) show the desired and the actual trajectory respectively in the x-y plane. In Figure 3(a) the desired trajectory starts at point A and follows a straight line to point B. Then it instantaneously jumps to point C and starts following an ellipse. In Figure 3(b), I denotes the initial location of the end-effector in the x-y plane. It takes a few seconds before the end-effector reaches the desired trajectory. Afterwards, the end-effector follows the trajectory remaining in some error bound. Figure 3(c) shows how the difference between the x-coordinates of the desired and the actual trajectories ($\texttt{diff}_\texttt{x}$) varies with time. After 20 time units of the start of tracking, there is a sharp rise in the tracking error. This is due to the discontinuity between the straight line and the ellipse in the desired trajectory. However, the end-effector corrects its trajectory very soon, and then follows the elliptical trajectory with a very small tracking error.

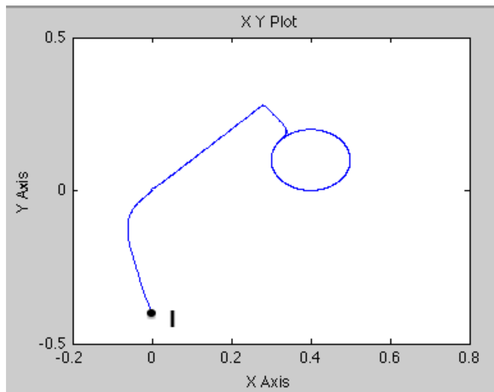
To ensure that *ModelRob Library* blocks perform the same functionalities as the blocks of the *Robotics Toolbox*, we perform a simulation based conformance testing. The conformance testing framework is shown in Figure 4. We feed the same input trajectory to the manipulator models implemented using *ModelRob Library* and *Robotics Toolbox*. Then we plot the difference between the x-coordinates and y-coordinates of the output trajectories for the two models with time. Figure 5 shows the difference between the x-coordinates of the output trajectories of the two models with time. The figure shows that the error is of the order of 10^{-15}, which provides confidence that the blocks in the *ModelRob Library* perform equivalent functionality with the corresponding blocks in the *Robotics Toolbox*.

We have used the RTW code generator to generate floating point C code from the robot model. The code generator has successfully generated more than 5000 lines of code in a few seconds. The code is modular, well-commented, and mostly easy to trace back with the model.

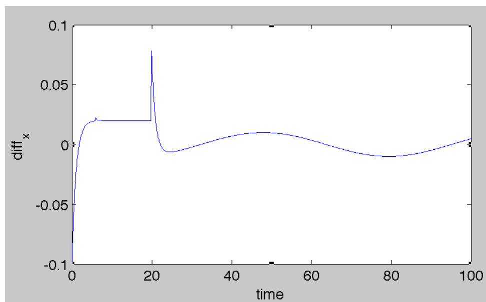
The Simulink blocks in our library by default have floating-point data types for the outputs of the blocks. A Fixed-point Simulink model can be generated from the floating-point model by appropriately selecting fixed-point data types for the outputs of different blocks. The Simulink blocks for trigonometric functions are replaced by the lookup table based implementation of the functions. We have developed two fixed-point models for circular trajectory — one is with 8-bit data width, and the other is with 16-bit data width. In the fixed-point models, the inputs are assumed to be coming from sensors whose outputs are of fixed length (8-bit or 16 bit). We compare the end-effector's actual trajectory for these fixed-point models with the end-effector's actual trajectory for the floating-point model. During simulation,



(a)



(b)



(c)

Fig. 3. (a) The desired trajectory of the end-effector (b) The actual trajectory of the end-effector (c) The difference between the x-coordinate of the desired and the actual trajectory with time

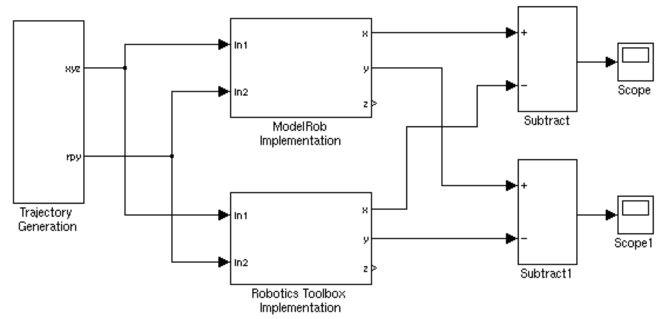


Fig. 4. The framework for conformance testing between the *ModelRob Library* model and the *Robotics Toolbox* model

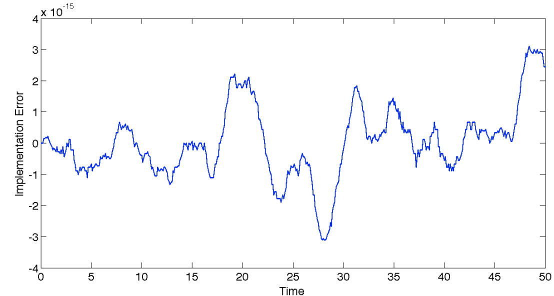
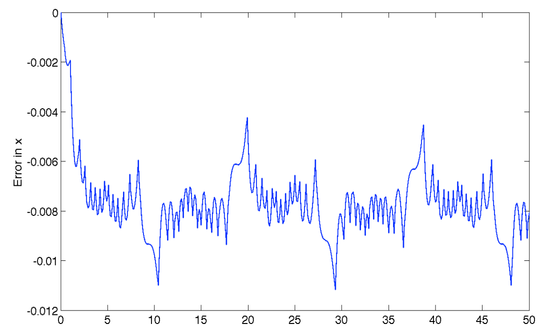
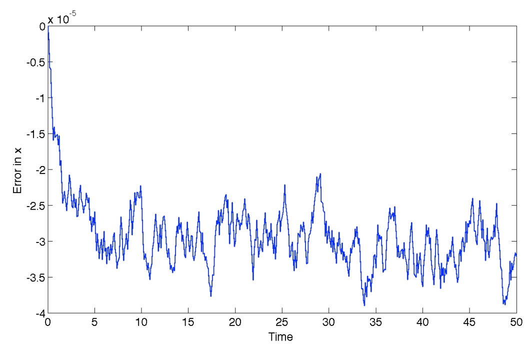


Fig. 5. The difference of the x-coordinates of the trajectories of the end-effectors in the model developed using *Robotics Toolbox* and *ModelRob Library*

at different time points we plot the difference between the actual x and y coordinates of the end-effectors in a fixed-point model and the floating-point model. Figure 6 shows how the difference between the x-coordinates of the floating-point Simulink model and a fixed-point Simulink model varies with time (Figure 6(a) is for the 8-bit model and Figure 6(b) is for the 16-bit model). The plots show that for the 8-bit model the magnitude of the difference is bounded by 0.012m, and it is 0.00004m for the 16-bit model, when we simulate the models for 50 time steps. The cost of a fixed-point processor increases with the number of bits. Thus, there is a trade-off between the accuracy of the output and the cost of the implementation.



(a)



(b)

Fig. 6. The difference of the x-coordinates of the trajectories of the end-effectors in the floating-point model and fixed-point (a) 8-bit model and (b) 16-bit model.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce *ModelRob*, a Simulink Library for modeling robot manipulator functionalities, and outline a model-based development process for robot manipulators based on Simulink modeling tool. We carry out a case study on Cartesian space motion control of a robot manipulator. We have modeled a manipulator controller using our library and successfully generated C code from the model. This shows the feasibility of model-based development of a robotic system using Simulink, a widely used tool in many embedded domains.

The model-based development process mainly comprises three key subprocesses: modeling, code generation and verification. In our case study we have concentrated on the first two subprocesses. Though a few verification tools for Simulink are available, none of the tools addresses the specific need of verifying properties of robot manipulators mainly because of the nonlinear behavior of the system. To fulfill the verification need in the robotics domain, we need verification tools capable of handling nonlinear behavior. Recent advancement in nonlinear constraint solving [35], [36] is expected to enable us getting progress in that direction.

The current version of *ModelRob Library* supports modeling of robot manipulators with only revolute joints, and excludes the block set required to model a manipulator with prismatic joints. The library currently does not include blocks to perform robot dynamics. There are other variants of stationary robots, for example, cylindrical robot, spherical robot and parallel robot; and mobile robots, for example, wheel robot and legged robot, which *ModelRob Library* currently does not support. The sole objective of this paper is to show that the model-based development approach can be successfully applied to the development of robot software, and to motivate development of exhaustive library to cover all variants of robots.

REFERENCES

[1] J. H. Craig, *Introduction to Robotics: Mechanics and Control*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[2] L. Sciavicco and B. Siciliano, *Modelling and Control of Robot Manipulators*. Springer, 1995.

[3] E. Papadopoulos and S. Dubowsky, "Coordinated manipulator/spacecraft motion control for space robotic systems," in *Proceedings of International Conference on Robotics and Automation*, 1991, pp. 1696–1701.

[4] X. Cyrill, G. J. Jaar, and A. K. Misra, "Dynamical modelling and control of a spacecraft-mounted manipulator capturing a spinning satellite," *Acta Astronautica*, vol. 35, no. 2/3, pp. 167–174, 1995.

[5] M. J. H. Lum, D. Trimble, J. Rosen, K. F. II, H. King, G. Sankarnarayanan, J. Dosher, R. Leushke, B. Martin-Anderson, M. N. Sinanan, and B. Hannaford, "Multidisciplinary approach for developing a new minimally invasive surgical robot," in *Proceedings of BioRob Conference*, 2006.

[6] G. Sankarnarayanan, H. King, S.-Y. Ko, M. J. H. Lum, D. C. W. Friedman, J. Rosen, and B. Hannaford, "Portable surgery master station for mobile robotic telesurgery," in *Proceedings of the First International Conference on Robot Communication and Coordination*, 2007.

[7] Y. Lingtao, Z. Lixun, and Z. Jiliang, "Research on clamping dexterity of manipulator and application of minimally invasive surgery robot," in *Proceedings of International Workshop on Intelligent Systems and Applications*, 2009, pp. 1–4.

[8] The MathWorks, "MathWorks^{TM} - Simulink Reference," http://www.mathworks.com/products/simulink/.

[9] The Mathworks, "Real-Time Workshop," http://www.mathworks.com/products/rtw/.

[10] dSPACE, "TargetLink," http://www.dspaceinc.com/en/inc/home/products/sw/pcgs/targetli.cfm.

[11] P. I. Corke, "A robotics toolbox for MATLAB," *IEEE Robotics and Automation Magazine*, vol. 3, no. 1, pp. 24–32, 1996.

[12] The Mathworks, "Writing S-Functions," http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sfunctions.pdf.

[13] C. Kapoor, "A reusable operational software architecture for advanced robotics," Ph.D. dissertation, The University of Texas at Austin, 1996.

[14] R. Gourdeau, "A robotics object-oriented package in C++," Ph.D. dissertation, École Polytechnique de Montréal, 2001.

[15] H. Bruyninckx, "Open robot control software: The OROCOS project," in *Proceedings of International Conference of Robotics and Automation*, 2001, pp. 2523–2528.

[16] M. Geisinger, S. Barner, M. Wojtczyk, and A. Knoll, "A software architecture for model-based programming of robot syatems," *Advances in Robotics Research*, pp. 135–146, 2009.

[17] S. Bamer, M. Geisinger, C. Buckl, and A. Knoll, "EasyLab: Model-based development of software for mechatronic systems," in *IEEE/ASME International Conference on Mechtronic and Embedded Systems and Applications*, 2008, pp. 540–545.

[18] J. F. Broenink, Y. Ni, and M. A. Groothuis, "On model-driven design of robot software using co-simulation," in *Proceedings of SIMPAR 2010 Workshops International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2010, pp. 659–668.

[19] C. Schlegel, T. Haler, A. Lotz, and A. Steck, "Robotic software systems: From code-driven to model-driven designs," in *Proc. 14th Int. Conf. on Advanced Robotics*, 2009.

[20] J. Shi, M. Törngren, D. Servat, C. J. Sjöstedt, D. Chen, and H. Lönn, "Combined usage of UML and Simulink in the design of embedded systems: Investigating scenarios and structural and behavioural mapping," in *Proceedings of the 4th workshop of Object-Oriented Modeling of Embedded Real-time Systems*, 2007.

[21] T. Farkas, C. Neumann, and A. Hinnerichs, "An integrative approach for embedded software design with UML and Simulink," in *Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2009.

[22] R. S. Hartenberg and J. Denavit, "A kinematic notation for lower pair mechanisms based on matrices," *Journal of Applied Mechanics*, vol. 77, pp. 215–221, 1955.

[23] The MathWorks, "Simulink Fixed Point," http://www.mathworks.com/products/simfixed/.

[24] R. Alur, A. Kanade, S. Ramesh, and K. C. Shashidhar, "Symbolic analysis for improving simulation coverage of Simulink/Stateflow models," in *Proceedings of Internation Conference on Embedded Software (EMSOFT)*, 2008, pp. 89–98.

[25] M. Balasubramanian, A. Bhatnagar, and S. Roy, "Tool for translating Simulink models into input language of a model checker," in *Proceedings of International Conference on Formal Engineering Methods*, ser. Lecture Notes in Computer Science, vol. 4260, 2006, pp. 606–620.

[26] The Mathworks, "Simulink Design verifier," http://www.mathworks.com/products/sldesignverifier/.

[27] C. Chen, J. S. Dong, and J. Sun, "A formal framework for modeling and validating Simulink diagrams," *Formal Aspects of Computing*, vol. 21, no. 5, pp. 451–483, 2009.

[28] P. Roy and N. Shankar, "SimCheck: An expressive type system for Simulink," in *Proceedings of the NASA Formal Methods Symposium*, 2010.

[29] N. Sharygina, J. Browne, F. Xie, R. Kurshan, and V. Levin, "Lessons learned from model checking a NASA robot controller," *Formal Methods in System Design*, vol. 25, pp. 241–270, 2004.

[30] D. Bhatt, S. Hickman, K. Schloegel, and D. Oglesby, "An approach and tool for test generation from model-based functional requirements," in *Proceedings of the International Workshop on Aerospace Software Engineering(AeroSE2007)*, 2007.

[31] Reactive Systems Inc., "Reactis," http://www.reactive-systems.com.

[32] T-VEC, "Simulink Tester," http://www.t-vec.com.

[33] K. J. Hayhurst and D. S. Veerhusen, "A practical approach to modified condition/decision coverage," in *Proceedings of 20th Digital Avionics Systems Conference*, 2001, pp. 14–18.

[34] *RTCA Inc., Document RTCA/DO-178B*, Federal Aviation Administration, Jan. 11, 1993, advisory Circular 20-115B.

[35] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, pp. 209–236, 2007.

[36] M. K. Ganai and F. Ivancic, "Efficient decision procedure for non-linear arithmetic constraints using CORDIC," in *Proceedings of Formal Methods in Computer-Aided Design (FMCAD)*, 2009, pp. 61–68.